NASA-CR-192740

NAGW-1333

CIRSSE DOCUMENT #42

TECHNICAL REPORTS

[illegible]

11 [illegible]

153730

p- 114

(NASA-CR-192740) THE ORGANIZER: PLANNING TASKS WITH AN EMERGENT CONNECTIONIST/SYMBOLIC SYSTEM (Rensselaer Polytechnic Inst.) 114 p

N93-71628

Unclas

Z9/63 0153780

CIRSSE

Center for Intelligent Robotic Systems for Space Exploration

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

THE ORGANIZER: PLANNING TASKS WITH AN EMERGENT CONNECTIONIST/SYMBOLIC SYSTEM

By:

M.C. Moed

Department of Electrical, Computer and Systems Engineering
Department of Mechanical Engineering, Aeronautical Engineering & Mechanics
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

September 1989

CIRSSE Document #42

THE ORGANIZER: PLANNING TASKS WITH AN EMERGENT CONNECTIONIST/SYMBOLIC SYSTEM

Michael C. Moed

Center for Intelligent Robotic Systems for Space Exploration (CIRSSE)
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

THE ORGANIZER: PLANNING TASKS WITH AN EMERGENT CONNECTIONIST/SYMBOLIC SYSTEM

Michael C. Moed

Center for Intelligent Robotic Systems for Space Exploration (CIRSSE)
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

Abstract

A methodology is proposed for the research and implementation of a learning and planning system for robotic tasks named *The Organizer*. Under the framework of the Organization level of the Intelligent Machine, the Organizer follows a bottom up approach by creating condition/action/effect rules through experimentation and observation of an abstracted environment. Created rules may have probabilistic effects in non-deterministic environments. Feedback is returned from the lower levels of the Intelligent Machine which provides a measure of complexity and describes the difficulty of executing a particular action in a given environment. The rules form a symbolic model of the effect of the Intelligent Machine on a given environment and are compiled into a connectionist-based complexity model to allow modeling and generalization of complexity values to untested rules. The rules are also compiled into a goal-directed Boltzmann Machine which allows subtask determination, skill formation and goal-directed exploration through the maximization of an analytic value representing Knowledge in the system. Methods are shown for overlaying existing symbolic learning systems on the Organizer's rule structure by a symbolic generalization example. A graph search planner is used to develop task plans which achieve a user-provided goal through the use of rules and skills. In the development of this architecture, comparisons are made between Procedural planners, Symbolic learning systems, Neural networks and Classifier Systems. Further research is proposed in the areas of goal-directed skill formation and planning.

Contents

List of Figures

1 Introduction

As technology progresses, man creates and explores worlds and environments which were previously unattainable or non-existant. With the last several years, man has ventured into space, explored the depths of the oceans, and charted frozen lands in the Antartic. Man's pursuit of knowledge has also led to the construction of environments required for nuclear energy production and its counterpart, nuclear waste disposal. Ironically, man has also discovered that he cannot survive unaided in each of these created or encounted environments because of inclement conditions or extreme hazards. However, man has also found it necessary to continue limited exposure to these new worlds in order to maintain currently required capabilities and pursue further research.

To eliminate man's exposure to such hazards, it is necessary to create a device which can perform with some of the anthropomorphic capabilities of humans, and substitute the device in place of a human operator in these environments. For situations in which decision making by a human is not available or expedient, these devices must also possess the necessary intelligence to perform their tasks reliably in a world which may possess varying degrees of uncertainty. These devices have been termed *Robots* or *Intelligent Machines*.

In general, intelligent robots or machines must be able to perform a variety of functions ranging from high intelligence to high precision tasks, depending on the situation. The structure of such a device can be stratified from high level mechanisms to low level functions. Some examples of high level mechanisms for an Intelligent Machine are:

- Learning about its capabilities and its environment through observation and discovery or through a teacher.
- Forming an abstract internal model of its environment based on what the Intelligent Machine has learned.
- Developing abstract plans and making decisions based on desired goals and needs through the use of internal models.

Some middle level mechanisms are:

- Selecting and scheduling planned tasks for execution.
- Mapping abstract concepts to current environmental situations.
- Aiding the formation of internal models by processing features of sensory information.
- Monitoring task execution and completion.

Finally, lower level functions include:

- Executing tasks with a dictated degree of precision.
- Obtaining raw sensor data and performing limited processing on it.
- Providing reflex responses in extreme situations.

Methodologies for achieving structures for Intelligent Machines have been proposed by [Alb75, Sar79, Vam87, APW88]. Saridis has proposed a mathematical framework for an Intelligent Machine which can be viewed as an intersection of the major disiplines of Artificial Intelligence, Operations Research, and Control Theory. Saridis' system is formulated to optimize a performance measure to allow for efficient planning and execution.

This paper addresses the problem of devising a framework for the high level mechanisms of an Intelligent Machine patterned after Saridis' model. These upper level functions form the Organization level of this model. Included in this work is a proposal for a suitable computational architecture which provides the necessary functionality for this level. Also included are proposals for detailed research into two parts of the architecture: goal-directed learning and exploration, and the development of abstract task plans based on learned internal models.

1.1 An Overview

Environments which are hazardous to human life provide a good application for Intelligent Machines. Valavanis [Val86] provided a case study detailing the performance of an intelligent robot in a Nuclear Power facility. An overview of this study will serve to outline the issues involved in designing the high level mechanisms of the machine.

1.1.1 The Scenario

Valavanis describes the following scenario:

Suppose that an emergency situation occurs in a Nuclear facility due to a power failure which renders normal electronic remote control of the plant valves impossible. As a result, the pressure control in this highly radioactive environment fails and a breach of primary containment is identified and operating personnel are evacuated.

Several valve, flange or pipe related operations must be performed to control the pressure in the facility, depending on the state of the plant when the power fails. An autonomous mobile robot with visual capabilities is used to perform these operations. The mobile robot has been trained to perform tasks in this type of environment, and has an internal model of its capabilities. After receiving a goal to be achieved from a human operator, it is the responsibility of this robot to enter the radioactive area, observe the state of the environment, develop a plan to achieve the goal based on its internal models, and execute each of the tasks in that plan. The robot is responsible for planning and performing actions towards the goal which are efficient and have high likelihood of success.

For example, suppose such an emergency situation occurs and the robot is given the command "Shut off Pipe 1". The robot, based on its past training and planning experience, might generate the following plan:

1. Robot Travel to Entrance
2. Robot Travel to Toolchest
3. Arm2 Open Toolchest
4. Arm1 Grasp Wrench
5. Robot Travel to Pipe 1
6. Arm2 Grasp Pipe 1
7. Arm1 Turn Valve with Wrench
8. Arm2 Release Pipe 1
9. Robot Travel to Toolchest
10. Arm1 Release Wrench in Toolchest
11. Arm2 Close Toolchest
12. Robot Travel to Exit

This plan would be generated if the Wrench is known to be in the Toolchest, and it is necessary to turn the Valve on Pipe 1 with the Wrench in order to shut off Pipe 1. It should be noted that if the Toolchest was Open upon entry to the situation, the step which opens it would have been omitted.

1.1.2 High Level Mechanisms

The following steps must be taken in order to allow the high level functions of the mobile robot to perform with the capabilities described above:

1. Training the mobile robot to allow for the development of internal models of objects which it may encounter and its capabilities in manipulating these objects.
2. Training the mobile robot to plan for goal-oriented behavior. This allows the formation of the proper sequence of valve, flange, pipe, manipulation and travel operations.

3. Receiving an operator goal and development of an abstract plan of tasks based on its internal model which minimizes both complexity and risk of failure.

The above scenario indicates several required high level functions and provides a brief introduction to the desired capabilities of an Intelligent Machine.

1.2 Functional Decomposition of High Level Mechanisms

Based on the above observations, we can segment the high level functions of the Organization level as follows:

- Training:
  - Training for internal model development: Through some procedure, such as supervised learning, reinforcement learning, or learning through exploration and discovery, the Intelligent Machine must build up an abstract internal representation of actions it can perform on objects. By abstracting the representation of an object from its actual form, many of the details of the object and its environment are lost or ignored. At this level, the abstracted representation allows the machine to develop conceptual relationships between actions and objects without bothering with complex details. Included in this representation should be the difficulty or complexity of performing the actions, in order to provide a measure of efficiency when executing a task.
  - Training for goal-seeking behavior: Using one of the learning methods above, the Intelligent Machine must learn the consequences of actions it performs on objects in its abstracted environment. The effects the Intelligent Machine has on objects may be probabilistic which facilitates the modeling of likelihood of success. By developing an abstract internal model of its effects on objects, it can formulate a plan of actions to execute in order to transform objects from an initial state to a desired goal state.
- Planning: Provided with a description of the current (or initial) state of objects in the abstracted environment, the Intelligent Machine must plan an ordered series of tasks to achieve a desired goal state for the objects. The following considerations should be made:
  - Since the high level functions maintain an abstract description of the objects in the world, the plan that is formulated is also abstracted to a large degree. The function of planning at this level is to develop an ordered list of tasks to perform which achieve an abstracted goal. The plan, which at minimum provides the necessary actions required to achieve the goal, ignores many environmental details. Lower levels in the system must map

this plan to a more detailed description of the environment. This mapping will enhance the goal directed behavior of the abstracted plan by adding additional subtasks dictated by a richer environmental model.

This abstraction can be illustratated using the mobile robot example above. In the generated plan, the intelligent robot has specified the steps:

1. Arm2 Open Toolchest
2. Arm1 Grasp Wrench

Within the Toolchest, the Wrench may be in various positions covered by several other tools. However, the environmental specifics of the situation are not richly represented in the abstraction space. Therefore, the plan does not include subtasks which solve the more detailed problems. These subtasks must be added to the goal plan by on-line planning mechanisms which are resident in the Intelligent Machine, but outside of the domain of the Organization Level.

– The plan should be formulated in a manner which minimizes the complexity of its execution while maximizing the likelihood of success. To do this, the planner must build on information stored in the internal models developed during training.

Provided with this introduction, it is possible to give the formal problem statement.

1.3 Statement of Problem

A computational architecture and its underlying analytic functions must be created for the Organization Level of the Intelligent Machine as described by Saridis. The Organization level must possess the following capabilities:

- The ability to build and store an abstract internal model of the effect of the Intelligent Machine on objects in its environment.
- The ability to represent the capabilities of the Intelligent Machine by modeling the learned complexities of actions executed by the Machine.
- The ability to exhibit goal-directed behavior through learning.
- The ability to analytically describe tasks in terms of complexities and likelihood of success.
- The ability to develop an abstract plan which optimizes an analytic criteria based on task complexity and likelihood of success which achieves a desired goal state from an initial state based on internal models.

1.4 Problem Domain

The desired capabilities of the Organizer dictate which internal models should be considered. Thus, it is important to consider the problem domain before developing the internal structure of the system.

As presented above, the Organization level must develop a sequence of tasks which form a plan or procedure by manipulating objects in the environment. The example of a mobile robot in a hazardous nuclear environment described typical responsibilities of this level, and demonstrated one practical application. Let us consider two types of planners, determine which is best for planning in the Organization level, and decide on a suitable data structure for the internal model.

The first type of planner is termed a Procedural Planner. Characteristics of a Procedural Planner are:

- Functionally, a generated plan calls for object manipulation which changes the state of the environment.
- Many types of objects are manipulated or transformed through a host of different actions.
- Objects change state in the environment but generally do not decompose into subobjects.
- A goal is achieved by developing an ordered sequence of different actions on objects wich transform the environment to a desired state.

Procedural planners use groups of individual states to represent objects in the environment. Examples of this type of planner are predicate calculus planners and expert systems.

The second type of planner is termed a Hierarchical Planner. Characteristics of a Hierarchical Planner are:

- Functionally, a generated plan calls for the composition or decomposition of objects.
- Objects can be formed by combining subobjects.
- Interrelations between object parts and whole is often detailed.
- A goal is achieved by executing tasks which reach a certain state or node in the hierarchical object model.

Hierarchical planners use hierarchical data structures such as graphs or trees to represent objects. Examples of this type of planner are assembly planners such as [HdMS88, HdM89, KM89]

From these descriptions, the Organization level best fits within the domain of Procedural planners. Therefore, the internal model of this level should be able to represent objects as sets of individual states in the environment.

1.5 Problems Addressed

The problems addressed in this research are:

- The design of a suitable computational architecture and internal models for the Organization level of the Intelligent Machine.
- The determination of an analytic criteria which combines complexity of task execution and likelihood of task success for use in goal-directed behavior.
- The development of methods for goal-directed exploration and learning which seek to optimize the analytic criteria. This is one functional block of the architecture.
- The development of an abstract planner which formulates a list of actions which change objects in the abstracted environment from an initial state to a goal state through the optimization of the analytic criteria. This is a second functional block of the architecture.

1.6 Organization of this Report

The report is organized as follows: Section 1 presents an introduction and overview of the problem. A literature search of Procedural Planners and Symbolic Learning Systems, Neural Networks, Classifier Systems and Saridis' Intelligent Machine concepts is presented in section 2. Also included in section 2 is a discussion on the method of approach for this research. Section 3 describes the architectural model for the Organization level, some preliminary results, along with a detailed description of the proposed research. Section 4 provides a list of the proposed research, and other research which should be performed under this architecture.

2 Literature Review and Method of Approach

The goal of this literature review is to provide the reader with background on both past and on-going research in the areas of planning and learning as they pertain to the development of the Organization level of the Intelligent Machine. To accomplish this goal, research from the fields of Procedural planners and Symbolic Learning Systems, Neural Networks, Classifier Systems and Saridis' Intelligent Machine concepts will be presented and critiqued. A method of approach will be formulated and contrasted with techniques presented in the review.

2.1 Literature Review

2.1.1 Procedural Planners

Procedural planners are planning systems characterized by the following features:

- Functionally, a generated plan calls for object manipulation which changes the state of the environment.
- Many types of objects are manipulated or transformed through a host of different actions.
- Objects change state in the environment but generally do not decompose into subobjects.
- A goal is achieved by developing an ordered sequence of different actions on objects which transform the environment to a desired state.

Some systems of this nature have also been called Domain Independent planners or Predicate Calculus planners. Most of these planners were developed in the field of Artificial Intelligence and used theorem proving by resolution as a method to create a plan which transformed an initial state into a desired goal state.

STRIPS [FN71] was one of the first AI planners. In STRIPS, the problem space is divided into the world model, a goal state, and a set of operators which acted on object states in the world model to produce new object states. The world model is created from a set of well-formed formulas (wff) which comprise the state of objects in the world. The operators are first-order predicate calculus rules which contain two parts: condition of use, and effect of use. The goal state is also a wff.

For example, the world state might be:

ON(BOX1,TABLE)
ON(BOX2,FLOOR)
AT(ROBOT,ROOM1)
GRASP(ROBOT,NOTHING)

A typical operator might be:

$ON(x,TABLE) \wedge AT(ROBOT,TABLE) \Rightarrow GRASP(ROBOT,x)$

The goal state might be:

$$ON(BOX2,TABLE)$$

Along with the effects of an operator is a list of wff to be added to the world model (called the *add list*) and a list of wff to be deleted from the world model (called the *delete list*) when the operator is executed. A search strategy named Means-End Analysis is used to eliminate differences between the present world model and the goal state by selecting subgoal operators. This is accomplished by fomulating a "difference clause" comprised of object states which are in either the present world model or the goal state but not in both. Operators to be applied are then selected based on whether their effect clauses remove difference clauses, or whether clauses on the add list of an operator's effect can be used to resolve-away difference clauses. This technique recursively breaks down the problem into subgoals which it attempts to solve through matching or resolution.

Since Means-End Analysis tries to solve many possible subgoals in order to solve a problem, a copy of working memory must be stored at each level of the search tree. Since the amount of storage required would be huge, STRIPS instead maintains a list of all wff for all possible configurations of world states in a global memory. Flags are set at each level of the search tree which indicate which clauses in the global memory are active at that node.

STRIPS has several problems. First, the Means-End Analysis search strategy is often not effective for plans which must take the world model further away from an initial state in order to reach a goal. Second, Means-End Analysis does not guarantee that a goal will be found if one exists. Third, when a plan is found from an initial state to a goal, it is not known how good the plan is, or if it is the most efficient plan. Fourth, STRIPS is a top-down approach to planning, since it requires a user to encode all the possible rules, conditions and effects that can occur, as well as all the possible world model states in global memory. This type of top-down approach suffers when the system encounters an unexpected situation in the real world which the user did not forsee and has not been exactly encoded in the rules. Fifth, STRIPS provides no learning or generalization capabilities.

In later work, [FHN72] MACROPS were added to the STRIPS system which allowed STRIPS to formulate and store generalized plans. However, the number of preconditons required to execute a MACROP were large so their usefulness was overshadowed by the amount of processing time required to test its applicability.

ABSTRIPS [Sar73] was built upon STRIPS and allowed the system to ignore a large number of operator preconditions during planning by the use of Abstraction Spaces. A plan was formulated at different levels of abstraction by the following procedure:

1. Rank the criticality of each precondition.
2. At a given abstraction space level, consider only the preconditions which have critical values equal to or above the value of that level.
3. Develop a set of tasks using Means-End Analysis to solve the problem at the current abstraction level.

The above procedure is repeated for each level of abstraction, progressing from the most critical levels to the least critical ones. At each stage, the more abstract plan is used to guide the lower, less abstract one.

The idea of a set of abstraction spaces can also be used to allow a planner to operate at different levels of detail, which is a large contribution of this work. However, since ABSTRIPS is built on the STRIPS system, it still contains many of the drawbacks native to STRIPS.

NOAH [Sar75] uses some of the methods developed in the ABSTRIPS system along with novel techniques to solve non-linear plans. This system enhances the Means-End Analysis search technique with a method for representing plans as non-linear sequences of actions and ordering these actions to achieve a desired goal.

NOAH maintains a hierarchy of representation spaces called a *procedural net*. Each node in the net represents an action which can recursively be broken down into a set of subactions, some of which can execute in parallel. As the net is expanded, a more detailed plan is formulated which achieves the goal. At each expansion level, a set of critics determine whether a parallel action node at that level is constrained by other parallel action nodes, and therefore must be sequenced in some manner. In this way, NOAH separates subgoals into parallel partial orderings of actions and then sequences these orderings to form non-linear plans.

A major problem with the NOAH system is its top-down approach which requires a vast amount of encoded knowledge in order to perform well. Not only does the user need to encode the conditions and effects of actions and the world model, but he/she must also encode the entire abstraction space. Many of the problems mentioned with STRIPS are also present here, such as evaluating the efficiency of a plan, and the guarantee of finding a plan if one exists.

MOLGEN [Ste81a, Ste81b] extends NOAH's capabilities by propagating constraints while formulating plans. Each task in the formulated plan posts constraints to a constraint list which cannot be violated by later tasks. The constraints serve to reduce the size of the search space by eliminating branches of the search tree which can no longer be pursued. The constraints are syntactically matched to outcomes of operators, and have no semantic base.

A large body of work has been done on symbolic learning systems. Based on predicate calculus rules, these learning systems create new rules or generalize existing ones to extend the knowledge in the system to new situations. The research on symbolic learning has extended to the domain of procedural planning systems, with studies focusing on new rule generation, and on composition of existing rules into skill sets for more efficient planning.

The ACT system [And83] is a symbolic learning system which operates on a rule database. It contains two subprocesses, *composition* and *proceduralization*. Composition takes a sequence of connected rules and combines them into a single rule or skill. Proceduralization generalizes existing rules by removing some domain-specific knowledge from the condition for rule execution.

Carbonell [Car86] divides search in problem solving methods into four classes, which are differentiated by the amount of domain-specific knowledge. These classes are:

- If no domain knowledge is available, weak methods such as heuristic search and means-end analysis must be used.

- If specific domain knowledge in the form of plans and procedures exists, these plans can be instantiated directly.

- If general plans apply but not specific ones, the general plans can be used to reduce the problem.

- If no specific plans apply, analogical transformations can be used from similar problems which have been solved previously.

Carbonell develops a process for reasoning by analogy called Derivational Transformation. This process examines problem steps, subgoals and decisions in order to recount past reasoning traces so they can be applied to new, similar problems.

DeJong [DeJ86] demonstrates the necessity of building schemata, or skills, into knowledge based systems in order to reduce the dimensionality of the search space for planning. This work focuses on creating and generalizing schemata, and classifies the situations for generalization as:

- Schema Composition: This operator is called on to combine known schema in a novel way. The technique is employed when one or more of the preconditions of a primary schema must be satisfied in a manner which has not yet been detailed.

- Secondary Effect Elevation: This procedure employs existing schema in a new way by modifying the existing schema to denote that they can be used under new situations.

- Schema Alterations: This technique modifies slots of a nearly correct schema so it can fit the requirements of a new situation.
- Volitionalization: This method incorporates a non-planning declaritive schema which provides factual information about a situation or event into a planning schema that can be used in problem solving.

DeJong uses each of these techniques for generalizing and creating schemata in order to reduce the number of search steps required for generating a task plan.

These examples provide a brief view of symbolic learning systems which operate on predicate calculus-based systems. Each system allows the creation or generalization of existing plans to new situations. However, it must be noted that these systems are based upon the top-down, user-encoded rules, so the systems will still contain and maintain the same inaccuracies present in AI procedural planners. Further, symbolic systems do not "compile" the rules in the data base to implicitly extract semantic relationships between symbols. Instead, these learning systems use symbolic substitution based solely on the symbol names. However, these symbolic learners could be layered upon other systems which provide semantic information about the symbols, and together form a unified learning and planning architecture.

2.1.2 Neural Networks

It is desirable for a high level planning system to possess or develop an internal representation of its actions and capabilities so that it can effectively issue commands which have a predictive outcome. Most robotic task planners [FN71, FHN72, Sar73, Sar75, Ste81a, Ste81b] attempt to achieve this representation through the use of syntactic models. These systems model the robot and its environment as a set of facts, and actions on the world as rules which manipulate the current set of facts or symbols. These strictly symbolic systems often suffer from Minsky's [Min61] "frame problem" which causes the system to perform unintelligent behavior due to the emergence of an unexpected set of conditions. Harnad [Har89] has charged that this problem is innate to such systems because the symbols or syntactics are not linked to actual meanings or semantics. This "Symbol Grounding Problem" also prevents the system from generalizing similar action/effects with repeated exposure because no semantic decomposition of a rule is available to explain the reason for its effects. The same limitation also inhibits a strictly symbolic system from predicting the effect of a newly created rule which has not yet been tested.

Harnad examines the use of a combined symbolic/connectionist architecture to solve the "Symbol Grounding Problem". In such a design, a neural network is used to ascertain the complex interrelationships between the symbols and their effects. The network can be used to direct and improve upon the functioning of the syntactic rules to allow for generalization of old rules and emergence of new ones.

Dethick and Plaut [DP86] examine the Physical Symbol System Hypothesis [NS76, New80] in light of recent advances in rule processing with distributed connectionist networks. By comparing the strengths and weaknesses of both systems, they conclude that a comprehensive theory of intelligence may require a hybrid model which combines the strengths of both approaches. Steels [Ste87] advocates self-organization of data through the use of neural networks as an alternative to totally explicit programming systems. Hutchison and Stephens [HS87] present the concept that symbolic rules are often only approximate descriptions of more complex or probablistic relationships in the real world. They claim that in large systems, the imprecision and context dependence of individual rules can produce unacceptable error rates in final outputs. In contrast, they assert that neural network systems allow much more precise representation of complex and imprecise relationships, and the knowledge can be learned directly from experience. The researchers concede, however, that the form of knowledge in distributed systems usually defies verbal description or explanation.

Neural networks, or *connectionist networks* generally refer to sets of simple processing units which are interconnected in a often complex way. The interconnecting links contain weights which determine the influence of the output of one processing unit on the input of another. Good reviews of connectionist models are provided in [AR88, RM86, Lip87]

The field of neural networks for computation arose from early work on simple binary thresholding units called the "McCulloch-Pitts" neuron [PM47]. This research developed methods for training a single neuron element to develop logical functions, such as AND or NOR

Rosenblatt's development of the Perceptron processing unit [Ros58] in which output of a unit is a sigmoidal thresholded linear function of the inputs to the unit, and use of groups of Perceptrons to learn some complex functions and classify data, furthered connectionist research. Widrow and Hoff [WH60] developed a technique for efficiently altering connection values, called "synaptic weights". Combined with the proof of the Perceptron Convergence Theorem [Blo62] methods were established for modifying weights in a network of Perceptrons in order to reduce classification error. However, as Minsky and Papert demonstrated [MP69], a single layer of Perceptrons can not handle many classification problems, such as EXCLUSIVE-OR or PARITY functions.

Kohonen [Koh72] and Anderson [And72] simulatenously published identical models for associative memory (or Content Addressable Memory). The "linear associator" model provides a linear model for processing elements, or neurons, which deviates from the binary representation developed in earlier models. The method for initializing these networks was based on Hebbian learning, assigning weights as a factor of input/output node correlation.

Based on this initial work, many other connectionist theories developed. Hopfield [Hop82] developed a theory for binary content-addressable memory (CAM) based on a network with time varying neuron values. This theory used an Energy analog to demonstrate how dynamic changes in node values approached the correct associative output of the network by minimizing the Energy in the network. Later [Hop84], neurons with graded response were used in CAM design. Grossberg [Gro88] approaches connectionist theory from a neurobiological viewpoint, and demonstrated that the Energy function in Hopfield's CAM formed a Lyapanov function which could be minimized. Hirsch [Hir87] examines convergence in the Hopfield model in some detail. McEliece et al [MPRV87] discuss the capacity of the Hopfield model.

Feldman and Ballard [FB82] extolled the use of distributed representations for storing data in a network. Barto, Sutton and Anderson [BSA83] demonstrated a reinforcement learning technique for pattern classifying neurons.

From this background, two major connectionist models evolved which are relevant to the problems addressed in this report. They are Backpropagation networks [RHW86a] and Boltzmann Machines [AHS85].

Backpropagation networks are created using layers of Perceptron elements. The researchers of these networks have pointed out that by using *hidden layers* of Perceptrons between input and output layers, arbitrary mappings can be made between input and output data. A technique for training these inherently feedforward networks was developed based on gradient descent of the error between the actual output of the network and the desired output. The error between these values is "backpropagated" from output nodes, to hidden nodes, to input nodes and modifies the synaptic weight between nodes at each layer. Simple Backprop networks can easily learn PARITY or EXCLUSIVE-OR operations.

Applications for Backpropagation networks are numerous and cover a broad range. The capability for creating arbitrary mappings between input and output values has led some researchers to use these networks as Transfer Functions from a control-theoretic viewpoint. Kawato et al [KUIS88, MKSS88] use Backpropagation networks to develop the multiplicative constants of non-linear terms for the inverse-dynamics of a Puma manipulator. Their model, which is based on the motor cortex in the brain, is able to produce a torque value given a desired angular change in the first three joints of the Puma over a set of trajectories. Goldberg and Pearlmutter [GP88] have used these feedforward networks to control a two-link direct-drive arm. Their network is trained by providing a window of trajectory points as input, and the correct torque as the desired output.

The ability to classify patterns is another important application of Backpropagation networks. Since so many experiments have been done under this application, only two are mentioned here for examples. Using a network, Marra et al developed a mechanism for terrain classification using texture for an ALV [MDM88]. Given a scene image in RGB, statisical measures for regions of the image are provided as input to the network. The network produces a classification of the region such as "road", "grass" or "sky". Ruck [Ruc87] used satellite range information to classify ground targets such as tanks, jeeps, and trucks using a Backpropagation network.

Boltzmann Machines are connectionist networks which use the minimization of an Energy formulation to produce its output. A Boltzmann machine contains a set of nodes which can take on binary values, and a set of weights which dictate connections between nodes. In a manner similar to Backpropagation networks, the nodes can be labeled as input nodes, hidden nodes, or output nodes. However, unlike Backprop, Boltzmann machine nodes are generally not arranged in feedforward levels.

The Energy in the Boltzmann Machine is a function of the node values and weights between nodes. It can be though of as a correlation value between node pairs which provides a measure of "goodness" for a particular input-output pair. The lower the Energy, the more likely the output response is the correct one for the particular input data. Given a particular input data set on its input nodes, a Boltzmann machine searches the set of possible states on the hidden and output nodes in order to find a set of node states which minimize the Energy in the network. This set of states is the correct associative output of the network.

The search technique generally used for determing the network output of a Boltzmann Machine is Simulated Annealing [KJV83]. Grossberg [Gro88] has shown that a Boltzmann Machine is identical to a Hopfield network which uses Simulated Annealing to alter the characteristics of its node thresholding functions. Methods have been developed to train a Boltzmann machine to associate input-output patterns [AHS85, Sus88].

Several systems have been developed which combine symbolic processing with neural networks to create knowledge-based or predicate calculus systems. Touretzky and Hinton [TH85] used a distributed Boltzmann Machine architecture to represent two types of production systems. The first system contains rules which consist of pairs of working memory triples for the rule condition, and an arbitrary set of triples which must be added to or deleted from working memory as the rule effect. Typical rules are of the form:

Rule-1: (F A A) (F B B) $\Rightarrow$ +(G A B) -(F A A) -(F B B)

The second production system is similar, but allows variable matching in the condition part of a rule. For example:

Rule-2: (x A B) (x C D) $\Rightarrow$ +(P D Q) -(R S T)

where x is a variable to be matched by working memory elements.

The weights of the network are fixed once setup by a user. These weights are used to represent the rules and working memory elements. Good results were obtained with a working memory alphabet size of 25 symbols, a set of about six rules, and six elements in working memory at a time.

Touretzky [Tou87] expanded the production system concept for neural networks by developing the DUCS architecture which provides multi-level distributed representations for frame-like concept structures. The goal of this research is to develop a powerful short term memory that can construct and manipulate concepts rapidly. Given some slot name/slot filler values as cues, DUCS can retrieve entire frames from concept memory. DUCS can also complete frames which have empty slot values.

For example, given the frame:

AGENT:	JOHN
VERB:	THROW
OBJECT:	x
DESTINATION:	FOX
LOCATION:	HOUSE

DUCS would retrieve the correct frame with x = ROCK.

The architecture of DUCS is based on the Hopfield and Tank model [HT85] which is an optimization network based on Hopfield's 1984 design. Again, all concepts are stored a priori by the user by designating the connection weights. Once these weights are assigned they are fixed and the network cannot learn new concepts. In other work, Hinton developed methods for learning some types of concepts [Hin86].

Dolan and Dyer [DD87] present the CRAM system which also performs role binding in knowledge schema. The procedural memory in this system is similar to that of Touretzky and Hinton, with the memory composed of many winner-take all cliques. Although they propose schema learning, they do not detail a technique for implementing it. Shastri [SA89] uses a high level representation of concepts to develop a connectionist system for rule based reasoning with multi-placed predicates and variables. This work focuses on the problem of variable binding in networks which try to perform predicate calculus operations. Again, this system possesses no ability to learn new rules or place bindings.

Day proposes a method for building an architecture in which connectionist and standard symbolic AI implementation techniques complement each other [Day87]. The theory behind the system calls for a connectionist network to observe the internal workings of a symbolic AI program and thereby learn to carry out the same problem solving behavior. Day proposes the use of a Backpropagation network to learn the AI rules. As he states, a major problem with this proposal is how to achieve the desired linkage between the two systems, so the network can observe the behavior of the rules. He does recommend the network learning be achieved by watching pre- and post-effects of the expert system chaining, where the pre-effects are the input and post-effects are the desired output of the network. In the paper, Day develops a rough architecture for this theory.

As demonstrated by this review, neural networks have been used to create learning systems which can extract relationships between input data items, and produce a desired output from data set correlations. Also, neural nets have recently been applied to concept storage in knowledge-based systems, but without significant learning capabilities. Some work has been proposed on combining these two capabilities into a single model for expert systems, but without actual results.

2.1.3 Classifier Systems

A novel approach to rule-based systems was presented by Holland [HHNT86] and is entitled *Classifier Systems*. This system combines a simple representational scheme with highly general learning mechanisms to create a parallel, multi-rule production system.

A basic classifier system contains a list of classifiers, a message list, an input interface and an output interface. A classifier is string of characters from the three-letter alphabet (0,1,#) and is divided into a condition part and an action part. The form of a classifier is:

$$C_1, C_2, \cdots, C_r/\mathrm{A}$$

where C_i is a condition field and A is the classifier action. An example of a classifier is:

010101##10/111100#110

A classifier is active when a message on the message list matches the condition part of the classifier. The # symbol is a don't care element in the condition part, and matches messages with either a 0 or 1 in that field. The basic execution cycle of a classifier system is given by Holland as:

1. Place all messages from the input interface on the current message list.

2. Compare all messages on the current message list to all conditions of all classifiers and record all matches.

3. For each set of matches satisfying the condition part of some classifier, post the message specified by the action part to a new message list.

4. Replace the current message list with the new message list.

5. Process the message list through the output interface to produce the system's current output.

6. Go to step 1.

In addition to the basic mechanisms of message matching and posting, classifier systems allow learning through the *bucket brigade* algorithm and by the *genetic algorithm*. The first algorithm allows competition between classifiers by apportioning credit to them on the basis of past usefulness to the system. This competition allows stronger classifiers to have a higher success rate at posting messages to the message list than weaker classifiers. The genetic algorithm provides a method for new classifier creation from building blocks of previously successful classifiers.

The *bucket brigade* algorithm is a process of alternate bidding and payback. When a message is posted to the current message list, all classifiers matching the message bid for the opportunity of posting their actions to the new message list. The bid a classifier can make is a function of the strength of the classifier, the specificity of the condition of the classifier, and the support of the classifier. The strength of a classifier is a measure of its past usefulness. The specificity of the condition allows classifiers which are more detailed to have a higher probability of being selected to post messages. In effect, this condition implements a default hierarchy, where more general matching classifiers are chosen only when specific ones are weak. The support parameter is dependent on the strength of the past classifiers which posted the given matched message to the current message list. To some degree, this assigns a weighting to messages on the current message list. A classifier is probablistically chosen to place its message on the new message list depending on its total bid.

When a winning classifier places its message on the new message list, its strength is decreased by the amount of its bid. At the same time, all classifiers which sent the message (in the previous time step) matched by the winner have their strengths increased by a fraction of the bid of the winner. In this way, strengths are propagated backward through a chain of classifiers. When messages are posted which satisfy a goal-state, a reward is given to all the classifiers which posted the messages. Eventually, this reward is past back through the bucket brigade to the chain of classifiers which "set the stage" for the goal-state. Over repeated trials, this chain gets reinforced as the classifier strengths increase. When a classifier repetitively bids and fails, its strength continues to decrease, since it receive no payback. When the strength drops below a given threshold, the classifier is either removed or replaced by a new classifier.

The *genetic algorithm* is a search technique orignally presented in [Hol75]. In the context of classifier systems, the genetic algorithm performs selective recombination of existing classifiers to create new ones. Pairs of strong classifiers are selected and combined to create "offspring" classifiers. These offspring are placed in the classifier system with a strength value which is the average value of its two parents.

Goldberg [Gol83] employed a classifier system to generate a rule set to cover the operating conditions of a gas pipeline. The results of this work demonstrated the existance of default hierarchies. Wilson's [Wil85] Animat system was the first to demonstrate the bucket brigade under infrequent payoff conditions. Sutton [Sut88] has classified the bucket brigade as a *temporal – difference method* and has begun placing the algorithm in a theoretical framework.

Wilson and Goldberg [WG89] present several inherent problems with classifier systems. They cite that a primary weakness of the bucket brigade algorithm is the difficulty in generating and maintaining long chains of rules which achieve a desired goal. This weakness is due to the fragility of long chains and the difficulty of reinforcing early rules in a chain.

Riolo [Rio87] showed that the bucket-brigade algorithm can correctly allocate strength when classifiers are coupled together to form chains. In [Rio89], Riolo examined the emergence of coupled chains of classifiers through use of the Triggered Chaining Operator and the genetic algorithm. In this experiment, it was determined that classifiers present in the early generations of the system may become parisites on chained classifiers and lead to the demise of the entire coupled sequence. To overcome the breakdown of coupled chains, Riolo recommends, among other things, that competition between coupled and uncoupled rules be biased so that coupled rules are usually executed to completion.

Wilson [Wil87] proposes the use of a hierarchical credit allocation scheme for maintaining long chains. He shows that using bucket brigade for reinforcement requires 150 repetitions of a 10 step sequence in order to produce proper reinforcement, which is quite inefficient. Instead, Wilson proposes using bucket brigade on short, behaviorial chains which are arranged hierarchically to form the complete task. A high-level task in the hierarchy is broken down using the bidding algorithm from standard classifier systems, and payment is distributed to all levels of the hierarchy.

Booker [Boo89] attempts to overcome long chain problems by representing each situation with a cluster of rules instead of a single classifier. The structure of a cluster represents regularities in the input categories that excite it, and the similarity that each of these rules has in obtaining a goal. In this work, Booker points out that using the genetic algorithm for rule discovery based on classifier strength is very inefficient, and leads to extended learning efforts.

Zhou [Zho87] separates groups of classifiers into context memories in order to develop systems which achieve context-dependent goals. By developing classifier context memories, he allows the classifier system to maintain strong rules and chains for each goal, such as finding a path through different mazes. A set of context classifiers is called into main memory when the maze which it operates on is presented as the problem to be solved. If different contexts are not used, chains which find paths through one experimental maze would become weak and die in another experimental maze. Since the strength of chains and rules in one context is not valid when another goal is sought, Zhou demonstrates the need to separate rule and chain strength from goal-directed behavior in classifier systems.

Classifier systems possess several interesting features. They are combined learning/planning systems which develop new rules and exhibit goal-directed behavior. Also, the representational simplicity of classifiers fosters the development of default hierarchies of general and specific rules, and provides simplistic mechanisms for rule specialization. Third, the strength of a rule provides a mechanism to evaluate performance of the system.

However, there are problems inherent to classifier systems. It is difficult to generate and maintain long chains of classifiers and this complicates planning. Second, using the genetic algorithm to generate new rules based solely on classifier strength is not efficient. Third, there is an exhibited need to separate how good a rule is (context independence) from how well it leads to a goal (context dependence).

2.1.4 Saridis' Intelligent Machine

Since 1977 Saridis has been developing an engineering approach to the design of an Intelligent Machine [SV88, Sar79, Val86, SM88, MS89]. This approach, called Hierarchically Intelligent Control, is designed to organize, coordinate and execute anthropomorphic tasks by a machine possessing various amounts of autonomy. This approach utilizes analytical (probablistic) models to describe and control the various functions of the Intelligent Machine.

The Intelligent Machine consists of the following three layers:

1. The Organization level.
2. The Coordination level.
3. The Execution level.

These layers are organized in a tree like structure are shown in Figure 2.1, and are arranged according to the **Principle of Increasing Precision with Decreasing Intelligence.** The function of each layer is described briefly [Sar89]:

1. The Organization Level is responsible for high level decision making. It must organize a set of abstract rules or primatives to perform goal planning tasks. It combines inductive reasoning and inference capabilities to formulate such task plans.

2. The Coordination Level is an intermediate structure serving as an interface between the Organization and Execution levels. It combines the commands from the Organzation level with real-time world information to generate a proper sequence of subtasks for execution by the Machine. [SG84, WS88]

3. The Execution Level performs the appropriate actions in the environment as dictated by the Coordinators. These actions can be expressed as control functions and a measure is assigned to determine the execution performance.

A performance measure is provided as feedback to each level of the Machine. This value provides evaluative information about the performance of that level, and allows that level to modify its future actions based on the the evaluation of past executions. For the Organization Level, this feedback allows for reinforcement learning of successful plans, and avoidance of disastrous actions. The Machine can therefore be derived analytically as an optimization problem designed to extremize the performance value at each level, as well as the performance of the Machine as a whole.

The focus of this approach is to provide an analytical structure to each level of the machine. To develop this analytical structure, physical values are assigned to abstract concepts from the field of Machine Intelligence in order to place the system within a mathematical framework. The analytic formation is based on the following fundamental definitions:

Def. 1: Machine Knowledge is defined to be the structured information acquired and applied to remove ignorance or uncertainty about a specific task pertaining to the Intelligent Machine.

This definition defines Machine Knowledge as a variable which can be assigned and examined in the Intelligent Machine. When executing a task, the amount of Knowledge in the Intelligent Machine changes. Therefore, we must also define the Rate of Machine Knowledge, which updates the cumulative Knowledge in the Machine.

Def. 2: Rate of Machine Knowledge is the flow of knowledge through an Intelligent Machine.

Assuming that the Intelligent Machine contains a database of rules, a mechanism must be defined which operates on this database in order to update the cumulative Knowledge in the Machine. This operator is called Machine Intelligence.

Def. 3: Machine Intelligence (MI) is the set of actions which operates on a database (DB) of events to produce flow of knowledge (R).

In the Organization level of the Intelligent Machine, a task plan to achieive a goal is developed by minimizing the uncertainty or complexity of the plan. This uncertainty is a function of the imprecision of the process to be executed. Similary, the complexity of a process is a function of the required precision. This is defined as:

Def. 4: Imprecision is the uncertainty of execution of the various tasks of the Intelligent Machine.

On the other hand, one may define Precision as follows:

Def. 5: Precision is the complement of Imprecision, and represents the complexity of a process.

The above definitions present an outline for the development of an analytic structure for an Intelligent Machine, but do not inherently provide the actual measures to be optimized at each level. Based on these definitions, Saridis developed an engineering analog to the problem formulation by casting the variables Knowledge, Rate of Machine Knowledge and Uncertainty in terms of physical quantities. These physical quantities are Energy and Entropy.

For the Intelligent Machine, Knowledge (K) about a particular state (n) in the system is defined as:

$$K(n) \doteq Energy$$

It is necessary to develop a concept which provides an analytic relationship between the knowledge possessed about a state in a system, and the uncertainty that the system is in that state given that knowledge. From Jaynes' principle of Maximum Entropy [Jay57], we can relate Knowledge of State n to the Probability that the System is in State n (abbreviated $P(K(n))$):

$$P(K(n)) = e^{-\alpha - K(n)}$$

where α is a probability normalizing constant to insure that

$$\sum_n P(K(n)) = 1$$

This probability distribution comes from the field of statistical mechanics, and relates the Energy of a system to the probability of the system being in the state with that Energy value.

From this definition, it is apparent that when $K(n)$ is extremized, the Probability that the System is in State n approaches 1. This indicates that possessing a large amount of Knowledge about a particular state increases the probability that the system can correctly recognize when it is in that state.

Uncertainty, which is equivalent to the imprecision of the state of the system, is a function of $P(K(n))$. An entropy measure is used to relate these two values:

$$H(K(n)) = -\sum_n [P(K(n)) ln\{P(K(n))\}]$$

Here, $H(K(n))$ is the Uncertainty that the System is in State n. Saridis has shown that Entropy can be used as a suitable measure to minimize in optimization problems in order to guarantee good performance.

The Rate of Machine Knowledge (R), which reflects the updating of Knowledge over time can be simply described as:

$$R = \frac{K}{T}$$

where T is a discrete time interval.

The advantage of using Saridis' Intelligent Machine as a blueprint for planning systems is that it provides an analytic framework to operate within. Using the details presented here, a planner can be created which develops task plans by minimizing the uncertainty of the plan, and can describe its actions analytically in terms of Machine Knowledge, Rate of Machine Knowledge, Machine Intelligence, Uncertainty and Complexity. This planner would form the Organization level of the Intelligent Machine.

2.2 Method of Approach

It is desired for the learning and planning system to have the following capabilities:

- The ability to build and store an abstract internal model of the effect of the system on objects in its environment.
- The ability to represent the capabilities of the system by modeling the learned complexities of actions executed by the robot.
- The ability to exhibit goal-directed behavior through learning.
- The ability to analytically describe tasks in terms of complexities and likelihood of success.
- The ability to develop an abstract plan which optimizes an analytic criteria based on task complexity and likelihood of success which achieves a desired goal state from an initial state based on internal models.

The AI predicate calculus-based procedural planners provide methods for developing an ordered set of tasks which achieve a given goal. To do this, they maintain an abstract internal model of their effects on the environment, but do not possess any learning capabilities. The user must encode all the possible condition/action/effect rules, but this may not adequately represent what the robot can actually do, as stated in the "frame problem". Also, these rules have deterministic effects, which is often an incorrect model of actual robot systems. These rules also contain no mechanism for representing the complexity of executing the rule in the given environmental situation.

Symbolic learning systems which are layered on these top-down planners can contain and maintain these pre-encoded problems. Symbolic learning systems also are not able to extract "semantic" relationships between the symbols, since the learning procedure incorporated in such systems is based on symbol matching.

Neural networks provide a suitable mechanism for learning relationships between symbols, thereby establishing a "semantic base", if the symbols are simple enough to be provided as input to the network. Boltzmann machines are also able to provide a quantitative performance value based on the Energy of the network, a feature necessary to provide analytic information about a plan. Although techniques exist for training connectionist networks and methods have been demonstrated for using neural networks for devising predicate calculus rules, very little work has been done on combining these two types of systems.

Classifier systems present a rule-based system which has the representational simplicity required by neural networks, and provides techniques for generalization and specialization of rules. However, goal-directed behavior through use of the bucket brigade algorithm is inefficient. Also, the genetic algorithm is not efficient for generating useful new rules by combining old rules strictly by rule strength.

The Organization level of Saridis' Intelligent Machine provides an analytic environment for the incorporation of the strengths of each of the above systems into a unified learning/planning model. The systems are incorporated in a bottom-up, emergent model as follows:

- Use the simpliticity of classifiers to represent rules which are in the form of condition/action/effect.

- Allow the rules to be created through the Machine's experimentation with the environment, and the observation of its effects.

- Allow a probability to be associated with the effect portion of a rule to incorporate the likelihood that the condition/action part leads to the given effect.

- With the execution of an action for a given condition, receive feedback from the Coordination level of the Intelligent Machine which provides the complexity of the task.

- Maintain a neural network model of complexities for condition/action pairs. A neural network allows generalization of complexities across similar condition/action pairs by developing an inherent relationship between the symbols.

- Maintain another neural network which extracts the relationships between conditions for an action and effects of the action. This network will build "semantic" relationships between symbols across a rule and should be used to foster emergent goal-directed behavior by providing information on objects which change state due to the action. Employ this neural network to develop emergent skill sets in order to minimize search during planning by using the Energy of the net as a performance measure.

- Use a search technique for formulating a set of tasks to a goal which minimizes the complexity of tasks while maximizing likelihood of success during execution.

This method of approach creates an emergent connectionist/symbolic system for planning robotic tasks within the framework of the Intelligent Machine. If desired, it is possible to overlay top-down symbolic learning techniques on the rules which emerge, but techniques for doing this will not be explicitly examined in this work.

3 Design and Operation of the Organizer

3.1 Introduction

The Organizer is responsible for high-level task planning and decision making in a combined planning, coordination and execution system. Together, the Organizer (task planner) and lower levels of the system (task coordination and execution) perform tasks to affect the environmental state within which the system is operating. To develop an engineering approach to the design of the Organizer, the functionality of this system has been patterned after the Organization level of the Intelligent Machine [SV88, Sar79, Val86, SM88, MS89].

3.1.1 Responsibilities of the Organizer

A list of features which encompass the Organizer are as follows:

1. The Organizer is responsible for generating high level plans and decisions.
2. A User must provide a description of the goal which the Organizer will attempt to achieve through planning.
3. The Organizer must have information about the state of its environment.
4. The Organizer must provide commands in a given grammar to the Coordination level for execution in the environment. The result of the execution of a command will be a change in the state of the environment.
5. Feedback from the lower system levels indicating the complexity/imprecision of executing a command must be available to the Organizer.
6. Over time, the Organizer must develop an internal model of its effects on the environment in order to facilitate the planning of tasks.
7. The Organzier must also develop an internal model of the capabilities of the execution system within which it operates.
8. The Organizer must provide an analytic measure of its performance. and make decisions based upon the optimization of this measure.

To accomplish these responsibilities, the Organizer is divided into 5 main units. These units are:

1. The Rule Store. This unit maintains the list of rules and their effects.
2. The Complexity Model. This unit stores the execution complexity of a particular sentence provided from the Organizer to the Coordination level and predicts complexity values for similar sentences.

3. The Generalizer. This unit generalizes rules held in the rule store to allow for a wider range of application.
4. The Boltzmann Machine for Directed Exploration. This unit allows goal-directed behavior in the Intelligent Machine.
5. The Planner. Given a goal, this unit employs the above four functions to construct a set of sentences which achieve that goal.

It is the purpose of this chapter to detail the functions of the units presented above.

3.2 Input/Output Description

To understand the design of the Organizer, the following input/output characteristics must be described which link the Organizer to the lower levels of the Intelligent Machine, as well as the Machine to its operating environment.

1. The state of the environment as it is perceived by the Organizer.
2. The nature of the environment within which the Machine Operates.
3. The output that the Organizer provides to the Coordinator level.
4. The effect of the lower levels on the environment.
5. The feedback provided by the Coordination level to the Organizer.

3.2.1 State of the Environment

The Organizer is intended to be a high level planner and handles data and commands in an abstract form. One reflection of the abstract nature of the data within the Organizer is the internal representation of objects in the environment.

The environment consists of a set of $\mathbf{R}$ objects, $\Omega = (\omega_0, \omega_2, \cdots, \omega_{R-1})$. Each object ω_j can be in any of $m(j)$ states. Let Q_j be a vector $Q_j = (q_j^0, q_j^1, \cdots, q_j^{m(j)-1})$ where $q_j^k \epsilon (0, 1)$ represents the states of object ω_j. Let $q_j^k = 1$ if object ω_j is in state k, else $q_j^k = 0$. Therefore, the vector Q_j is a binary string which indicates the *inclusion* or *exclusion* of a particular state for object ω_j. This mapping from the actual environment to a binary state string is performed by the lower levels of the Intelligent Machine and is provided to the Organizer.

The state of the environment is denoted by $\mathbf{Q} = \Gamma_{i=1}^{n} Q_i$ where $\mathbf{\Gamma}$ is the string concatenation operator. $\mathbf{Q}$ is therefore a binary string consisting of the states of all objects in the environment. The length of $\mathbf{Q}$ is given by $\mathrm{M} = \sum_j m(j)$.

For example, the object *bottle* may be in state $(full, \; halffull, \; empty)$. It may also be in the state $(on-table, \; in-cabinet, in-gripper)$. This would be represented by a binary string of length six. Assume that *bottle* is object number b. Then:

$\omega_b = bottle.$
$Q_b = (q_b^0, q_b^1, q_b^2, q_b^3, q_b^4, q_b^5).$
$q_b^0 = 1$ if *bottle* is *full*, 0 otherwise.
$q_b^1 = 1$ if *bottle* is *halffull*, 0 otherwise.
$q_b^2 = 1$ if *bottle* is *empty*, 0 otherwise.
$q_b^3 = 1$ if *bottle* is $on - table$, 0 otherwise.
$q_b^4 = 1$ if *bottle* is $in - cabinet$, 0 otherwise.
$q_b^5 = 1$ if *bottle* is $in - gripper$, 0 otherwise.
For simplicity, the state string **Q** can also be indexed as $Q = (q_0, q_1, \cdots, q_{M-1})$.

Some parallels can be drawn between this representation scheme and methods used in other research. Classifier systems [HHNT86] use binary strings as messages to trigger internal and external events. This binary message string is also used to represent the quasimorphism structure which describes objects and their features in the environment in which the Classifier operates.

3.2.2 Nature of Operating Environment

It is important to consider the nature of the environment within which the Intelligent Machine operates in order to construct an Organizer which will perform well. For this system, we assume that the environment as perceived by the Organizer has the following features:

1. The environment can change slowly over time without influence from the Intelligent Machine.

2. The environment is not completely observable.

These features combine to force a non-deterministic structure on the design of the Organizer. Since the environment can change slowly, the effects of actions by the Intelligent Machine on objects may also change slowly. This forces the Organizer to monitor the execution of actions that it plans in order to continually update its selection scheme for future plans.

Since the environment perceived by the Organizer is an abstraction of the actual environment, all the details which are present in the world are not available to the Organizer. This indicates that under certain contexts, the effects of an action in a particular environmental state may not be deterministic. In order to compensate for a not completely observable environment, a probablistic structure should be used to capture unmodeled relationships in the world.

3.2.3 Organizer Commands to the Coordinators

The Intelligent Machine consists of various effectors and sensors. The Machine can perform a set of actions which alter the state of objects in the world. The Organizer must be able to specify a command in a known grammar to the Coordination level directing the use of the sensors and effectors to change the state of the environment. From this description we can say:

$\mathbf{A} = (a_0, a_1, \cdots, a_r)$ is the set of effectors/sensors which are called *actors*.

$\mathbf{V} = (v_0, v_1, \cdots, v_s)$ is the set of *actions*.

$\mathbf{D} = (d_0, d_1, \cdots, d_t) = (\Omega \cup \emptyset)$ is the set of *direct objects*.

$\mathbf{I} = (i_0, i_1, \cdots, i_t) = (\Omega \cup \emptyset)$ is the set of *indirect objects*.

$\mathbf{S} = (\mathbf{A}, \mathbf{V}, \mathbf{D}, \mathbf{I})$ is the set of possible output *sentences* from the Organizer to the Coordination Level.

A particular sentence is given by $s_i = (a_j, v_k, d_l, i_m)$ where $(a_j \epsilon A, v_k \epsilon V, d_l \epsilon D, i_m \epsilon I)$

Then a sentence s_i is a 4-tuple set consisting of one actor, one action, zero or one direct objects, and zero or one indirect objects.

Let $\mathbf{N} = \|\mathbf{A}\| + \|\mathbf{V}\| + \|\mathbf{D}\| + \|\mathbf{I}\|$. Thus, $\mathbf{N}$ is the sum of the number of actors, actions, direct objects and indirect objects defined for the Organizer.

This sets $\mathbf{A}, \mathbf{V}, \mathbf{D}, \mathbf{I}$ form the *primative events* described in [Val86] However, the form of the grammar is an extension of the *repeatable* and *non − repeatable* events also described in that work.

Let $\mathbf{S}^*$ be the set of finite strings over $\mathbf{S}$ plus the null string, $\emptyset$. Then a plan $\mathbf{P}$ is any subset of $\mathbf{S}^*$.

An example of a sentence is $(ARM1\ MOVE\ PEN\ TABLE)$. In this example, $ARM1 \ \epsilon\ \mathbf{A}$, $MOVE \ \epsilon\ \mathbf{V}$, $PEN \ \epsilon\ \mathbf{D}$, and $TABLE \ \epsilon\ \mathbf{I}$. Note that $(PEN, TABLE)$ $\epsilon\ \Omega$ as well.

3.2.4 Effect of Lower Levels on the Environment

After the Organizer generates a plan $\mathbf{P}$, it is sent to the Coordination level for execution. Each sentence $s_i \ \epsilon\ \mathbf{P}$ is executed individually. The execution of s_i causes the environment to change state in some fashion, and the abstraction of the state can be observed by the Organizer. After the execution of sentence s_i, the Organizer perceives the environment to change from state $\mathbf{Q}$ to $\mathbf{Q}'$. Therefore, as perceived by the Organizer, there exists a transition function Λ: $\mathbf{S} \times \mathbf{Q} \rightarrow \mathbf{Q}$. The following assumptions are also made which are accounted for in the design of the Organizer:

1. Given a sentence s containing $d_j \ \epsilon\ \mathbf{D}$ and $i_k \ \epsilon\ \mathbf{I}$, with $(d_j = \omega_j,\ i_k = \omega_k) \ \epsilon\ \Omega$, and a probability function $\mathbf{P}^s_{\mathbf{Q}}(\mathrm{i})$ defined as the probability that bit i of state vector $\mathbf{Q}$ inverts due to the execution of s, we assume that $P^s_Q(i|i\epsilon Q_j \cup Q_k) \gg P^s_Q(i|i\epsilon\neg(Q_j \cup Q_k))$.

2. Let $\Delta(x, y)$ be the Hamming distance between two binary strings x and y. Given a sentence s which causes the Organizer to perceive a change of state from $\mathbf{Q}$ to $\mathbf{Q}'$, then $\Delta(\mathbf{Q}, \mathbf{Q}') \ll \mathbf{M}$.

The first assumption states that the execution of a sentence sent from the Organizer to the Coordination level tends to effect the state of objects which are directly mentioned in the string rather than other objects in the environment. This forces a *focus of attention* or *locality of effect*. The second assumption affords some degree of continuity to the actions of the Intelligent Machine by insuring that the environment does not alter radically due to the execution of a sentence.

3.2.5 Feedback Provided by the Coordination Level to the Organizer

Besides abstracting the state of the environment into a string **Q**, the Coordination level is responsible for providing a feedback response $\mathbf{H_c(s)}$ to the Organizer which indicates the complexity or difficulty of executing a particular sentence s. The feedback from the lower levels of the system should influence the formulation of a plan **P** to optimize this feedback response. Since Saridis [Sar88] was able to reformulate the system control problem to use entropy as a control measure, entropy can effectively be used as feedback from the Coordination level to represent the complexity or difficulty of executing s.

We must assume that there is some cost of determining the feedback value by the Coordination level each time a sentence s is passed from the Organizer. This cost is due to the computational complexity of determining $\mathbf{H_c(s)}$ by analytic formulation of the Coordinator, or due to the cost of feedback after executing the sentence s by the execution level. Since optimal plan formulation requires a complexity measure for each sentence in **P**, the Organizer should be able to internally develop a computationally efficient model of the feedback response for a given sentence. This would reduce the overall cost of executing a plan by the Intelligent Machine by removing the cost of interaction with the Coordination level during planning.

3.3 Internal formulation of the Organizer

With the input/output requirements of the Organizer detailed, it is possible to proceed with the description of the internal formulation of the Organizer. The following section provides a blueprint for the basic mechanisms which form the Organizer architecture, the methods for training the Organizer, and how the formulation of plans are facilitated by more complex structures.

The internal formulation can functionally be divided into training operations and planning operations. A brief description to these operations is provided here as an introduction to more lengthy definitions.

The main algorithm for non-goal directed training is named **PLAY**. PLAY creates rules by generating somewhat random sentences, sending the sentences to the Coordination level for execution, and observing the abstracted effects of the sentences on the environment once execution has completed. A rule is created by concatenating the state string of the abstracted environment (condition), the sentence string that was executed (action), and the new state string of the environment after the effects of the sentence have taken place (effect). Since the nature of the operating environment allows non-deterministic effects for particular condition and action pairs, a variable is maintained in the rule which stores the probability of the rule effect occuring given the condition and action pair.

Each rule also maintains a variable which stores the complexity of executing the sentence given the current environment (precondition). This value is computed by the Coordination and Execution levels of the Intelligent Machine and is supplied as feedback to the Organizer. New rules created through PLAY are added to the **Rule Store**.

A mechanism called The **Generalizer** is included in the architecture and provides one form of symbolic learning. This mechanism removes unnecessary preconditions of rules, so they can match novel situations. This extends the knowledge base represented by the rule store, by allowing application of existing rules to different conditions.

When a rule is generalized, it is necessary to compute the complexity of the generalized rule under a novel situation. Also, when the nature of the environment or the capabilities of the Intelligent Machine are altered, the complexity of executing a sentence might change. A model must be maintained in the Organizer which extracts the semantic relationships between symbols in rules in order to compute complexity of sentence execution under a given set of environmental conditions. This connectionist model is called the **Complexity Model**.

As demonstrated by the literature review, planning is a search process which is computationally expensive due to the size of the state space. In order to reduce the size of the search space, the Organizer must be able to build up skills from subsets of tasks which move toward a user-defined goal. While many approaches to skill or schema formation have been based on top-down symbolic learning approaches which require large amounts of a priori heuristic knowledge, it is desirable for the Organizer to develop some of this capability from its own emergent knowledge base. To do this, another connectionist network called the **Goal-Direct Boltzmann Machine** is trained on the rules in the rule store to develop semantic relationships between rule conditions and effects. After training this network, skills can be formed by presenting the Organizer with problem subgoals, and using the network to extract small chains of low complexity rules which achieve the subgoal. These chains form skills which can be added to the rule store and used in planning.

Planning involves the proper selection of rules and skills from the rule store which cause the abstracted environment to transform from an inital state to a goal state. Recognizing that the conditions and effects of rules are isomorphic to nodes in a graph, and sentences are graph arcs, the **Planner** uses a graph search technique to find a low complexity, high likelihood of success text of sentences which achieve the goal state. This text is then transferred to the Coordination level for plan execution.

In short, the Organizer follows the following procedure:

1. Training: Using the PLAY algorithm, experiment with the actual or simulated environment: PLAY builds an initial base of rules describing the capabilities of the Intelligent Machine.

- Generate sentences and send to Coordination level.
- Observe the probablistic effect of sentences on the environment.
- Form rules about the abstract environment.
- Receive a measure of sentence complexity from the lower levels of the system.
- Store these feedback measures in the Complexity model.

2. Knowledge abstraction: Expand the capabilities of the Organizer rules by abstracting knowlege:
   - Generalize rules to apply to similar, but untested cases.
   - Store rule effects in the goal-directed Boltzmann Machine.
3. Goal training: Generate skills through use of the goal-directed Boltzmann Machine.
   - Receive a goal state from a user.
   - Provide desired state changes as input to goal-directed Boltzmann Machine.
   - Discover low Entropy rules which move toward goal.
   - Store the sequences of rules and effects in the skill store.
4. Planning: Use a search technique to find a path from an initial state to a goal state.
   - Receive a goal state from a user.
   - Search the rule and skill store for a path of sentences from the initial state to the goal state which minimizes cost.
   - If the cost of the cheapest path is excessive, either fail or enter goal-directed exploration.
   - If the cost of the cheapest path is acceptable, send the text of sentences to the Coordination level and observe execution.

A picture of the system and its environment is provided in Figure 3.1 with appropriate command and feedback paths.

3.3.1 Basic Mechanisms and Operation

The basic mechanisms of the Organizer are:

1. The rule store.

2. The complexity model for sentence execution.

These mechanisms and their method of operation allows the Organizer to build up a rich knowledge about the capabilities of an Intelligent Machine. Together, these units provide both syntactic and semantic systems which allow the analytic formation of optimal plans based on the minimization of a cost criteria when combined with the advanced mechanisms of the Organizer. The next sections describe the function of each unit, and how it operates.

The Rule Store

The rule store contains a list of condition/action/effect triples and forms the main syntactic mechanism in the Organizer. The following definitions are needed:

Let $\mathbf{R} = (\Xi, \Sigma, \Upsilon, \mathbf{H}, \mathbf{P}_\Upsilon)$ define the *rule store*. The rule store is then the set of 5-tuples where Ξ represents the set of *conditions*, Σ is the set of *binary sentences*, and Υ is the set of *effects*. The value $\mathbf{H} \geq 0$ is the feedback entropies from the Coordination level for a particular state/binary sentence pair. The values $0 \leq \mathbf{P}_\Upsilon \leq 1$ represent the probability of the condition/action pair causing the given effect. The assignment of these values will be discussed in depth later in another section. A particular *rule* R_i is in the rule store if $R_i \in \mathbf{R}$.

Each $\Xi_i \in \Xi$ is a string $(\xi_i^0, \xi_i^1, \cdots, \xi_i^{M-1})$ where ξ_i^j is $\in$ (0,1,#) and is called a *field*. Note that the length of Ξ_i is M, the length of the state string Q. If a particular field in a string Ξ_i is 0, that field is said to be *excluded* from the condition. If a field is 1, that field is said to be *included* in the condition. If a field is #, that field is said to be *don't care*.

Similarly, each $\Upsilon_i \in \Upsilon$ is a string $(v_i^0, v_i^1, \cdots, v_i^{M-1})$ where $v_i^j \in$ (0,1,#). Note that the length of Υ_i is M, the length of the state string Q. If a particular field in a string Υ_i is 0, that field is said to be *excluded* from the effect. If a field is 1, that field is said to be *included* in the effect. If a field is #, that field is said to be *dependent on the condition*.

Each $\Sigma_i \in \Sigma$ is a binary string $(\sigma_i^0, \sigma_i^1, \cdots, \sigma_i^{N-1})$ where N is the sum of the number of actors, actions, direct objects and indirect objects defined for the Organizer. If a field in Σ_i is 1, that field is said to be *included* in the binary sentence. Else, the field is *excluded* from the binary sentence.

The binary sentences Σ are a direct one-to-one and onto mapping of the sentences S. Σ can be separated into four disjoint sets, each corresponding to the actor, action, direct object and indirect object sets of S. Let $\|\cdot\|$ denote the number of fields in a string. The first $\|\mathbf{A}\|$ fields of a binary sentence Σ_i are mapped to the actors, the next $\|\mathbf{V}\|$ fields of Σ_i are mapped to the actions, etc. If a particular field σ_i^k $(k \leq \|\mathbf{A}\| - 1)$ is set to 1, then actor a_k is represented in Σ_i. Note that exactly one actor, one action, zero or one direct objects and zero or one indirect objects is included in any Σ_i.

For example, given the following sets:

$\mathbf{A} = (ARM1,\ ARM2)$

V = $(MOVE, GRASP, RELEASE)$
D = $(BLOCK, BALL, TABLE)$
I = $(BLOCK, BALL, TABLE)$

Then the sentence $s_i = (ARM1\ MOVE\ BLOCK\ TABLE)$ would be mapped to the binary sentence σ_i = (10100100001). Here, $ARM1$ is in the sentence so the first field of the binary sentence is included. Since $ARM2$ is not in s_i, the second field of the binary sentence is excluded, etc.

For convenience, rules will appear as:

$$\xi\xi\cdots\xi/\sigma\sigma\cdots\sigma/vv\cdots v$$

such as:

$$100\#0100\#/100010010010/\#00\#00010$$

A rule is *active* if all of its conditions are *matched*. The conditions of a rule R_i are matched if for a given environmental state Q, the Hamming distance $\Delta(\Xi_i, \mathbf{Q}) = 0$. Note that the Hamming distance between a *don't care* field, #, in Ξ_i and a 1 or 0 in the corresponding field in **Q** is 0. The *active set* is the set of active rules.

A rule can *fire* when it is active. If a rule is chosen to fire, the binary sentence portion of the rule is mapped back onto the sentence grammar defined over S. The sentence is then passed to the Coordination level for execution.

The structure of the rules is very similar to the rules used in Classifier Systems. Both use a trinary algebra (0,1,#) for representing condition strings. Rules in both systems contain a sentence portion, which in Classifier systems is called the rule message. The rules are also similar to those in most forward chaining expert systems such as OPS5 [BFKM86]. In both Classifiers and forward chaining systems, the rules attempt to match a condition portion with a current state message, and then produce an action to the system or environment as output. However, the Organizer rule also contains an effect portion which provides a model of the environmental state after execution. Because of this portion, the results of a rule firing are known a priori to its execution. This allows for efficient planning, as will be described later in this work.

The Complexity Model

As discussed previously, the Coordination level of the Machine provides a feedback value to the Organizer which reflects the complexity/difficulty of executing a sentence passed from the upper to lower levels. This feedback value is called $\mathbf{H_c(s)}$ and is the entropy of executing the actions called for by sentence s in the current environment. This value is computed by the lower levels either through analytic means at the Coordination level, or by control performance at the Execution level. To minimize the cost involved with this computation, it is necessary to model in the Organizer the value $\mathbf{H_c(s)}$ due to the execution of sentence s in the current environment. This model should function as both 1) a database for recalling complexities of past state/sentence pairs; and 2) a predictor of complexities for state/sentence pairs not yet executed.

This model performs the "compatability evaluation" described in [Val86]. It produces a measure of complexity for binary sentences based on co-occurance probabilities of primative events developed over repeated presentations.

The predictive nature of this model will allow the Organizer to discard certain classes of sentences which it has learned perform pooly over past experience. This generalization across similar sentences requires some semantic knowledge about the co-ocurrance of actor, action, direct objects and indirect objects. Based on these criteria, a connectionist structure will be used to form the complexity model.

For input, the state of the abstract environment and the binary sentence will be place on the nodes of the network. The Energy of the network will then be calculated, which is equivalent to the trained complexity of the sentence in the given environment.

Let $\mathcal{B}_c = (\mathcal{L}, \mathcal{W})$ be a connectionist network. Let node levels $\mathcal{L} = (L_Q, L_{QQ}, L_A, L_{AV}, L_V, L_{VD}, L_D, L_{VI}, L_I)$. Each node level is defined as follows:

$L_Q = (n_Q^0, n_Q^1, \cdots, n_Q^{M-1})$ where **M** is the length of state vector **Q**.

$L_{QQ} = (n_{QQ}^{01}, n_{QQ}^{02}, \cdots, n_{QQ}^{ij})$ where $(i < j < \mathbf{M} - 1)$. Note: The number of nodes in L_{QQ} given by $\|\mathbf{QQ}\| = \frac{(M)*(M-1)}{2}$.

$L_A = (n_A^0, n_A^1, \cdots, n_A^{\|A\|-1})$ where $\|\mathbf{A}\|$ is the number of actors.

$L_{AV} = (n_{AV}^{01}, n_{AV}^{02}, \cdots, n_{AV}^{ij})$ where $(i < \|\mathbf{A}\|,\ j < \|\mathbf{V}\|)$. Note: The number of nodes in L_{AV} is denoted $\|\mathbf{AV}\|$ and equals the number of actors multiplied by the number of actions.

$L_V = (n_V^0, n_V^1, \cdots, n_V^{\|V\|-1})$ where $\|\mathbf{V}\|$ is the number of actions.

$L_{VD} = (n_{VD}^{01}, n_{VD}^{02}, \cdots, n_{VD}^{ij})$ where $(i < \|\mathbf{V}\|,\ j < \|\mathbf{D}\|)$. Note: The number of nodes in L_{VD} is denoted $\|\mathbf{VD}\|$ and equals the number of actions multiplied by the number of direct objects.

$L_D = (n_D^0, n_D^1, \cdots, n_D^{\|D\|-1})$ where $\|\mathbf{D}\|$ is the number of direct objects.

$L_{VI} = (n_{VI}^{01}, n_{VI}^{02}, \cdots, n_{VI}^{ij})$ where $(i < \|\mathbf{V}\|,\ j < \|\mathbf{I}\|)$. Note: The number of nodes in L_{VI} is denoted $\|\mathbf{VI}\|$ and equals the number of actions multiplied by the number of indirect objects.

$L_I = (n_I^0, n_I^1, \cdots, n_I^{\|I\|-1})$ where $\|\mathbf{I}\|$ is the number of indirect objects.

For simplicity, the nodes can also be indexed $\mathcal{L} = (n_0, n_1, \cdots, n_{\|\mathcal{L}\|})$ where $\|\mathcal{L}\| = \mathbf{M} + \|\mathbf{QQ}\| + \|\mathbf{A}\| + \|\mathbf{AV}\| + \|\mathbf{V}\| + \|\mathbf{VD}\| + \|\mathbf{D}\| + \|\mathbf{VI}\| + \|\mathbf{I}\|$.

Any node, $n_i \in (0, 1)$. The value of n_i is called its *activation*.

$\mathcal{W}$ is called the *weight matrix* for $\mathcal{B}_c$. $\mathcal{W}$ is of size $(\|\mathcal{L}\| \times \|\mathcal{L}\|)$. Each element $w_{ij} \in \mathcal{W}$ is called a *weight between node i and node j*. The following rules hold for weights in $\mathcal{W}$:

1. $\|w_{ij}\| \leq 1.0$.

2. Some weights are fixed at 0.0. Other weights can vary between -1.0 and 1.0.

3. If w_{ij} is fixed at 0.0, there is no connection between nodes n_i and n_j. Else, a connection exists between nodes n_i and n_j.

For the network $\mathcal{B}_c$, the following weight assignments are made:

1. For any nodes n^i_X and n^j_X, w_{ij} is fixed at 0.0 for $X \epsilon (Q, QQ, A, AV, V, VD, D, VI, I)$. This indicates that there is no connection between any two nodes on the same level.
2. For any nodes n^i_Q and n^j_{QQ}, w_{ij} is fixed at 0.0.
3. For any nodes n^i_A and n^j_{AV}, w_{ij} is fixed at 0.0.
4. For any nodes n^i_V and n^j_{AV}, w_{ij} is fixed at 0.0.
5. For any nodes n^i_V and n^j_{VD}, w_{ij} is fixed at 0.0.
6. For any nodes n^i_D and n^j_{VD}, w_{ij} is fixed at 0.0.
7. For any nodes n^i_V and n^j_{VI}, w_{ij} is fixed at 0.0.
8. For any nodes n^i_I and n^j_{VI}, w_{ij} is fixed at 0.0.
9. All other weights can vary between -1.0 and 1.0.

A diagram of this network is provided in Figure 3.2.

A connection between nodes n_i and n_j is *active* if w_{ij} is not fixed at 0.0, $n_i = 1$, and $n_j = 1$.

As part of the Intelligent Machine, the network is defined in terms of the analytic model. The output of $\mathcal{B}_c$ is defined as the *Execution Complexity of a Task given the State of the Environment* and is abbreviated $\mathbf{H_n}(\mathbf{s_i})$ for sentence s_i. Therefore, we define the $\mathbf{H_n}(\mathbf{s_i}) = \sum_{i=0}^{\|\mathcal{C}\|} \sum_{j=i}^{\|\mathcal{C}\|} w_{ij} n_i n_j$. This is similar to the energy definition for Boltzmann Machines defined in [HS86]

Training the complexity model

The method for adjusting $\mathcal{W}$ over repeated trials is called *training the network*. The following procedure is used to train the network to produce the complexity of a binary sentence Σ_i given Σ_i and $\mathbf{Q}$ as input. We assume that $\mathbf{H_{max}}$ is the maximum complexity value and $\mathbf{H_{min}}$ is the minimum value.

Given a state vector $\mathbf{Q}$, a binary sentence Σ_i, and a measure of complexity from the Coordination level, $\mathbf{H_c}(\mathbf{s_i})$, where s_i is a sentence mapped from the binary sentence Σ_i:

1. For each field q_i in $\mathbf{Q}$, if $q_i = 1$ then set $n^i_Q = 1$.

2. For each field σ_i in Σ_i where $(0 \leq i \leq \|\mathbf{A}\| - 1)$, if $\sigma_i = 1$, then set $n_A^i = 1$.
3. For each field σ_i in Σ_i where $(\|\mathbf{A}\| \leq i \leq \|\mathbf{V}\| - 1)$, if $\sigma_i = 1$, then set $n_V^i = 1$.
4. For each field σ_i in Σ_i where $(\|\mathbf{V}\| \leq i \leq \|\mathbf{D}\| - 1)$, if $\sigma_i = 1$, then set $n_D^i = 1$.
5. For each field σ_i in Σ_i where $(\|\mathbf{D}\| \leq i \leq \mathrm{N} - 1)$, if $\sigma_i = 1$, then set $n_I^i = 1$.
6. For each pair of nodes n_Q^i and n_Q^j $(i < j)$, if both nodes equal 1 then $n_{QQ}^{ij} = 1$.
7. For each pair of nodes n_A^i and n_V^j, if both nodes equal 1 then $n_{AV}^{ij} = 1$.
8. For each pair of nodes n_V^i and n_D^j, if both nodes equal 1 then $n_{VD}^{ij} = 1$.
9. For each pair of nodes n_V^i and n_I^j, if both nodes equal 1 then $n_{VI}^{ij} = 1$.
10. Set all other nodes to 0.
11. Compute the Complexity from $\mathcal{B}_c$, $\mathbf{H_n}(\mathbf{s_i}) = \sum_{i=0}^{\|\mathcal{C}\|} \sum_{j=i}^{\|\mathcal{C}\|} w_{ij} n_i n_j$.
12. Compute $\delta\mathbf{H} = \mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i})$.
13. For each and every non-fixed $w_{ij} \in \mathcal{W}$: If $\delta\mathbf{H} > 0$,

$$\delta w_{ij} = \alpha(1.0 - w_{ij}) \frac{\delta\mathbf{H}}{\mathbf{H_{max}} - \mathbf{H_{min}}} n_i n_j$$

If $\delta\mathbf{H} < 0$,

$$\delta w_{ij} = \gamma(1.0 + w_{ij}) \frac{\delta\mathbf{H}}{\mathbf{H_{max}} - \mathbf{H_{min}}} n_i n_j$$

Where $0 \leq \alpha, \gamma \leq 1$.

Each iteration of this algorithm is a *training step*.

Over repeated iterations, the output of the network $\mathbf{H_n}(\mathbf{s_i})$ should converge to $\mathbf{H_c}(\mathbf{s_i})$. The following theorem is useful:

Theorem 3.1. For a connectionist network of type $\mathcal{B}_c$ with a training procedure given above, let l equal the number of active connections in $\mathcal{B}_c$ for a given sentence s_i. If $\alpha < \frac{\delta\mathbf{H}}{l}$ then $\|\delta\mathbf{H}\|$ before a training step is larger than $\|\delta\mathbf{H}\|$ after a training step. Similary, $\gamma < -\frac{\delta\mathbf{H}}{l}$.

Proof: Let us first examine the case where $\delta\mathbf{H} > 0$. In this case, $\mathbf{H_c}(\mathbf{s_i}) > \mathbf{H_n}(\mathbf{s_i})$. For the updating step, we have:

$$\delta w_{ij} = \alpha(1.0 - w_{ij}) \frac{\delta\mathbf{H}}{\mathbf{H_{max}} - \mathbf{H_{min}}} n_i n_j$$

The maximimum value $(1.0 - w_{ij})$ can obtain is 2.0, since $-1.0 \leq w_{ij} \leq 1.0$. The maximum value $\frac{\delta\mathbf{H}}{\mathbf{H_{max}} - \mathbf{H_{min}}}$ can obtain is 1.0. Thus $\delta w_{ij} \leq 2\alpha$. Then, the maximum change in the output of the network due to a training iteration is $2 \times l \times \alpha$, since only l weights are updated.

We must show that $\|\mathbf{H_c}(\mathbf{s_i}) - (\mathbf{H_n}(\mathbf{s_i}) + 2l\alpha)\| < \|\mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i})\|$.

We know $\mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i}) > 0$. Assume $\mathbf{H_c}(\mathbf{s_i}) - (\mathbf{H_n}(\mathbf{s_i}) + 2l\alpha) > 0$. Then we must show that $\mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i}) - 2l\alpha < \mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i})$. This is true if $2l\alpha > 0$ which is always the case.

Now assume $\mathbf{H_c}(\mathbf{s_i}) - (\mathbf{H_n}(\mathbf{s_i}) + 2l\alpha) < 0$. Then $-\mathbf{H_c}(\mathbf{s_i}) + (\mathbf{H_n}(\mathbf{s_i}) + 2l\alpha) > 0$. For $-\mathbf{H_c}(\mathbf{s_i}) + (\mathbf{H_n}(\mathbf{s_i}) + 2l\alpha) < \mathbf{H_c}(\mathbf{s_i}) - \mathbf{H_n}(\mathbf{s_i})$ to be true, $\alpha < \frac{\delta \mathbf{H}}{l}$. The proof is analogous for γ.

The above updating scheme and supporting theorem provide a method for successive adaptation of the output of the network to mimic the feedback signal provided by the Coordination level. This adaptation is similar to gradient descent of the error for a given sentence s_i by modifying the weights of the network.

The constraints on α and γ given by the above theorem are somewhat restrictive. It is possible that the training routine will reduce the output error using larger values of α and γ. The important consideration is the distance of the current weight value to the shunting value (-1.0 or 1.0) which may account for a smaller output change, and allow larger constant values.

Extracting the complexity of a sentence

During execution, it becomes necessary to determine the complexity/entropy of a sentence without going through an entire training step. To extract the complexity of a given sentence s_i, one must first map the s_i to the binary sentence Σ_i. Then, for a given state vector Q, the complexity of s_i can be found by executing steps 1-11 in the training algorithm. The complexity, or entropy of s_i equals $\mathbf{H_n}(\mathbf{s_i})$.

A priori inhibitory connections and weights

It may be desirable to inhibit the execution of certain tasks or sets of tasks based on a priori knowledge about the nature of the tasks under particular environmental conditions. The Machine user should be able to encode this "innate" knowledge into the complexity model.

Assume we have two nodes n_i, n_j which should not be asserted together (i.e inhibited). The user should set $w_{ij} = 1.0$ and not allow this weight to be modified during training. By fixing this weight at 1.0, the complexity of any task requiring both nodes active will be high.

Discussion of the complexity model

With the formal model defined, a short discussion is necessary to provide intuitive insight to the model. The complexity model is based heavily on the Boltzmann machine architecture, and uses the energy function to extract the output value of the network. The values of the nodes can be either 1 or 0, meaning on or off. For this example, the weights ranged from -1.0 to 1.0. To maintain uniformity with other parts of the Organizer, it is necessary to change the structure so that the weights range from 0.0 to 1.0

The node levels of the model are segmented in a way which allows an easy mapping from the current state vector and a chosen sentence to the network. Each node in level L_Q is assigned the corresponding bit value of state vector Q. Similary, the node in L_A is asserted which corresponds to the actor in the given sentence, the node in L_V is asserted which corresponds to the action in the given sentence, etc. These nodes are called the *visible* nodes of the model.

The levels L_{QQ}, L_{AV}, L_{VD}, and L_{VI} allow the assertion of pairwise combinations of states, actions and actors, actions and direct objects, actions and indirect objects, respectively. These nodes allow the machine to form boolean functions between nodes. It is important to note that these nodes provide more than just the "AND" function. Through repeated training, the combination of these paired nodes with nodes of the other levels allows the formation of such boolean functions as "XOR", "NOR", etc. These nodes are called the *hidden* nodes of the model.

Similar to a proportional controller, the weights of active connections are trained to reduce the error between the current output of the network, and the desired output of the network. The theorem presented states that an individual training step will not increase this error for a particular sentence. This is important for stable operation. The idea behind this training technique is that over repeated presentations, certain active connections will always lead to very good or very bad values of complexity. These links will tend to shift toward -1.0 or 1.0 respectively. If the weights are restricted to the range of 0.0 to 1.0, the links will shift in a similar manner to these bounds. Other links will tend to oscillate in the middle since the output of the network when they are active ranges over the complexity scale. The effect of these links tend to die out over repeated presentations. In other words, the network self-organizes to find pairwise node combinations which tend to lead to bad performance, and ones that lead to good performance, and these nodes become the major influencing factors in calculating the network output.

It is important to see why this technique is used for training as opposed to one like backpropagation [RHW86a, RHW86b] or the Boltzmann machine training algorithm [AHS85]. Backpropagation requires an a prioi knowledge of a particular string, and the output of the network given that string as input. All of these paired values must be known before the net is trained. The same type of knowledge is required by the Boltzmann training technique. In contrast, the complexity model learns through repeated experience over the course of execution of the Organizer, and does not need to know all value pairs ahead of time (which is especially useful, since these values are not available). Also, the technique used to train the complexity model is called "Reinforcement Learning", because we are training a network response by providing a reinforcement signal. Backpropagation and the Boltzmann training technique both use "Supervised Learning". A detailed description of the differences between supervised and reinforcement training methods is provided in [Wil88]. Therefore, it can be trained with or without a teacher.

An example of a complexity model

To demonstrate the effectiveness of this model, several simulations are presented. The environment consists of a table and two objects labeled OBJ1 and OBJ2. The Machine consists of a robot arm and gripper which can either pick an object up or put an object down (GRASP or RELEASE). In the simulations, the Machine must learn the following:

1. It is easy to grasp an object when nothing is in the gripper.
2. It is easy to release the object which is in the gripper.
3. It is difficult to pick up any object when one is already in the gripper.
4. It is difficult to release an object unless it is in the gripper.

The model was built as follows:

1. The L_Q level consisted of 4 nodes:
   (a) $n_Q^0 = OBJ1\ in\ GRIPPER$
   (b) $n_Q^1 = OBJ1\ on\ TABLE$
   (c) $n_Q^2 = OBJ2\ in\ GRIPPER$
   (d) $n_Q^3 = OBJ2\ on\ TABLE$
2. The L_A level consisted of 1 node: $n_A^0 = ARM$.
3. The L_V level consisted of 2 nodes:
   (a) $n_V^0 = GRASP$
   (b) $n_V^1 = RELEASE$
4. The L_D level consisted of 2 nodes:
   (a) $n_D^0 = OBJ1$
   (b) $n_D^1 = OBJ2$
5. For this case, the L_I level was unnecessary and not modeled.

Including the paired levels (L_{QQ}, L_{AV}, L_{VD}) the network consists of twenty-one nodes and eighty-four modifiable connections. The weight matrix is assigned as discussed in the previous section. Each element in $\mathcal{W}$ was initialized to 0.0.

In the first simulations, the feedback $\mathbf{H_c}(s)$ is 1.0 for difficult tasks (tasks 3,4 above) and 0.0 for easy tasks (tasks 1,2 above). The goal of training is to match the output of the network with the provided feedback for every valid state/sentence combination.

Figures 3.3a - 3.3i show the output of the network for several test sentences over 200 training steps with α, $\gamma = 0.05$. Figures 3.4a - 3.4i show the output with α, $\gamma = 0.15$.

Determination of $\mathbf{H_n}(s)$ required 190 additions. Each training iteration required update of the 84 weights. After computing δH, each weight update requires one addition/subtraction and one multiplication.

The simulation results demonstrate that the network is able to learn the correct feedback values during the limited number of presentations. It is important to note that 200 presentations represent the total number of sentence instances the net trains on. In other words, for 200 presentations, each of the twelve valid sentences is presented to network an average of 16 times. This is a very small number of presentations when compared to techniques like backpropagation.

As shown in the simulations, with α, $\gamma = 0.05$, the network is able to provide correct responses within 5 percent to the test cases in about 165 training steps. This corresponds to about 14 presentations of each possible sentence to the network. With α, $\gamma = 0.15$, the network performs much better. It responds within 3 percent to the test cases in about 100 training steps. This averages to about 8 presentations of each sentence to the network. After 120 presentations, the test case response is about 100 percent accurate.

The second simulation demonstrates a case in which non-binary responses are provided as feedback. In this case, we try to model OBJ2 as a heavier object then OBJ1. To represent this, the feedback provided when grasping OBJ2 is not 0.0, but 0.2. Again, the simulation results show in Figures 3.5a - 3.5i that the network is successfully able to mimic the feedback values. After 120 training steps, or about 10 presentations of each of the twelve sentences, the output of the network is within 5 percent correct for each of the test cases. After 140 steps, it is within 2 percent of the correct values.

Operation of the Basic Mechanisms

The rule store and complexity model form the basic mechanisms of the Organizer. These mechanisms can be viewed as the primary building blocks required to provide syntactic and semantic structure to the system. It is within these mechanisms that most of the knowledge within the Organizer is learned and maintained. The next section describes the operation of these building blocks, and how they work together to form a base from which plans can be constructed.

PLAY Algorithm (Non-Generalized Rules)

The purpose of the rule store is to develop a database of condition/effect relationships for sentence execution by the Intelligent machine. Each rule $R_i \epsilon \mathbf{R}$ defines the effects Υ_i of executing sentence s_i when the environment is in a state which maches Ξ_i. The question arises: How is **R** built?

Intertwined with the rule store is the complexity model $\mathcal{B}_c$. For a given sentence s_i in state Q, $\mathcal{B}_c$ must develop a representation for the feedback complexities $\mathbf{H_c(s_i)}$ provided by the Coordination level. The question arises: How are relevant matched sentence/state pairs presented in order to effectively train $\mathcal{B}_c$?

These questions are answered by an algorithm called *PLAY*. PLAY forces the development of rules in the rule store, and provides legitimate sentence/state pairs for the complexity model. Through interaction with an abstracted non-deterministic environment, PLAY allows the development of rules which have probablistic effects and form a robust syntactic environmental model.

PLAY proceeds in this manner: Under the current abstracted environmental state, the Organizer randomly picks a sentence to execute. The probability that a given sentence is selected may be influenced by the user of the Machine. The sentence is then passed to the Coordination level of the Intelligent Machine and is executed in the environment, if possible. The Organizer receives the new abstracted environment string and the complexity of execution as feedback. If necessary, a new rule is formed by concatenating the precondition (current abstracted environmental string), the binary sentence and the effect (new abstracted environmental string). The complexity value of the rule is assigned and the probability of the precondition/sentence causing the given effect is updated. PLAY is described in more detail as follows:

Given an initial environmental state Q, a rule store R, a complexity model $\mathcal{B}_c$, a maximum complexity threshold θ_H, a new rule generation probability p_n, a set of probability density functions P_A, P_V, P_D, P_I which are the probabilities of selection a particular actor, action, direct object and indirect object, respectively, for a newly formed sentence, and P_R, the probability density function of selecting a particular rule from the active set:

1. Find the active rule set R_{active}: $(\forall R_i \epsilon \mathbf{R} : \Delta(\mathbf{Q},\Xi_i) = 0)$. If the active rule set is empty, go to step 2a.

2. Randomly select a value p in (0..1). If $p \leq p_n$ a new rule is generated as follows:

   (a) Generate a particular $a_n \epsilon A$ according to P_A.

   (b) Generate a particular $v_n \epsilon V$ according to P_V.

   (c) Generate a particular $d_n \epsilon D$ according to P_D.

   (d) Generate a particular $i_n \epsilon I$ according to P_I.

   (e) s_n is formed by the set (a_n, v_n, d_n, i_n).

(f) Map s_n to the binary string Σ_n.

(g) Transfer the sentence s_n to the Coordination level for execution.

(h) Receive $\mathbf{H_c(s_n)}$ as feedback from the Coordination level.

(i) Find the subset of rules in R_{active} which have the binary sentence Σ_n: $R'_{active} = (\forall R_j \in R_{active} : \Sigma_j = \Sigma_n)$.

(j) If $\mathbf{H_c(s_n)} > \theta_H$ the sentence could not be executed. Update the complexity model and set $\mathbf{H} = \mathbf{H_c(s_n)}$ for all $R_j \in R'_{active}$. Go to step 2.

(k) If $\mathbf{H_c(s_n)} \leq \theta_H$, the sentence was successfully executed. Observe the abstracted state vector $\mathbf{Q}'$, update the complexity model, and set $\mathbf{H} = \mathbf{H_c(s_n)}$ for all $R_j \in R'_{active}$.

(l) If $\forall R_j \in R'_{active}$, $\Delta(\Upsilon_j, \mathbf{Q}') \neq 0$, then create a new rule R'_n by concatenating $\mathbf{Q}$, Σ_n and $\mathbf{Q}'$. Assign $\mathbf{H} = \mathbf{H_c(s_n)}$ for R'_n. Else, let $R'_n = (R_j \in R'_{active} : \Delta(\Upsilon_j, \mathbf{Q}') = 0)$.

(m) Update the effect probability values $\mathbf{P}_\Upsilon$ for all rules $R_j \in R'_{active}$ and compute $\mathbf{P}_\Upsilon$ for R'_n.

(n) If new, add R'_n to $\mathbf{R}$.

3. If $p > p_n$, an existing rule is selected from $\mathbf{R}$:

(a) From R_{active} randomly select a rule R_s according to P_R.

(b) Map Σ_s to s_s. Transfer s_s to the Coordination level for execution.

(c) Receive $\mathbf{H_c(s_s)}$ as feedback from the Coordination level.

(d) Find the subset of rules in R_{active} which have the same binary sentence as R_s: $R'_{active} = (\forall R_j \in R_{active} : \Sigma_j = \Sigma_s)$.

(e) If $\mathbf{H_c(s_s)} > \theta_H$ the sentence could not be executed. Update the complexity model and set $\mathbf{H} = \mathbf{H_c(s_s)}$ for all $R_j \in R'_{active}$. Go to step 2.

(f) If $\mathbf{H_c(s_s)} \leq \theta_H$, the sentence was successfully executed. Observe the abstracted state vector $\mathbf{Q}'$, update the complexity model, and set $\mathbf{H} = \mathbf{H_c(s_s)}$ for all $R_j \in R'_{active}$.

(g) If $\forall R_j \in R'_{active}$, $\Delta(\Upsilon_j, \mathbf{Q}') \neq 0$, then create a new rule R'_n by concatenating $\mathbf{Q}$, Σ_s and $\mathbf{Q}'$. Else, let $R'_n = (R_j \in R'_{active} : \Delta(\Upsilon_j, \mathbf{Q}') = 0)$.

(h) Update the effect probability values $\mathbf{P}_\Upsilon$ for all the rules $R_j \in R'_{active}$ and compute $\mathbf{P}_\Upsilon$ for R'_n.

(i) Assign $\mathbf{H} = \mathbf{H_c(s_s)}$ for R'_n.

(j) If new, add R'_n to $\mathbf{R}$.

4. Let $\mathbf{Q} = \mathbf{Q}'$.

5. Go to 1.

Updating the effect probability values

The PLAY algorithm is responsible for updating the effect probability values $\mathbf{P}_\Upsilon$ for the all $R_j \in R'_{active}$. These values represent the probability of a particular Ξ and Σ pair causing a specified effect in Υ. The probability updating method can be likened to an attempt to revise a prediction of the outcome given a particular input to a system and a particular action by the system. This type of probability modeling scheme has been dealt with extensively in the field of Stochastic Learning Automata, and provides one method for updating the effect probability values.

Fu and his colleagues [Fu71, FL69a, FL69b, Fu67] were among the first to introduce stochastic automata to the control literature. Excellent reviews of the field of Stochastic Learning Automata are available in [NT74, ME70]. Borrowing from these theories, the probability updating scheme is:

Given $\forall R_j \in R'_{active}$, the active rule R'_n, $(R_j \neq R'_n)$, and some constant $0 < \mu < 1$:

1. If R'_{active} is empty then $P_\Upsilon = 1$ for R'_n. Else execute 2,3.
2. For each R_j, let $\mathbf{P}^{\mathbf{j}}_\Upsilon = \mathbf{P}^{\mathbf{j}}_\Upsilon$ - $\mu\mathbf{P}^{\mathbf{j}}_\Upsilon$.
3. For R'_n, let $\mathbf{P}^{\mathbf{n}}_\Upsilon = \mathbf{P}^{\mathbf{n}}_\Upsilon + \sum_{j \in \|R'_{active}\|} \mu\mathbf{P}^{\mathbf{j}}_\Upsilon$.

This technique is known as the Linear Reward-Penalty scheme (denoted L_{R-P}). It subtracts probabilty from rules which have effects that did not occur, and adds the total subtracted probability to the effect probability for a rule in which the effect did occur. The L_{R-P} scheme is known to be *expedient*. As shown in [CS67], L_{R-P} schemes work well in non-stationary environments (ones in which effects of actions can change over time). Other methods which are ϵ-optimal tend to lock onto certain actions, and lose their ability to change.

Other approaches may be used to update the effect probability values. Stochasic Approximation is also very appropriate to this type of model. Frequency of occurance is another method which may be suitable for particular implementations of the Organization level.

Updating the complexity model

The PLAY algorithm calls for the complexity model to be updated with each tested sentence. This is accomplished using the complexity model training method given previously, by providing the current binary sentence and Q as input, and adjusting the weights according to the feedback response from the Coordination level.

A priori rules

The design of the Organizer allows the user to encode a priori rules and place them in the rule store, if desired. If the pre-encoded rules have incorrect actions, they will perform poorly in experimentation and their effect probabilities will decrease. If the pre-encoded rules have correct actions, they can be used in planning along with rules developed during PLAY.

Discussion of basic operation mode

The basic operation mode described above by the PLAY algorithm allows the Organizer to build a large database of valid rules which can be applied to the environment. The rules in the rule store are very similar to those found in expert systems or predicate calculus planners with three important caveats. First, the rules in the rule store model the abstracted environment by direct interaction with the environment. A user does not need to enter the rules by hand in order to create the functions of the system. This eliminates Minsky's [Min61] "frame problem", which is caused when user-defined rules do not handle unexpected situations; on the contrary, in the Organizer system, the rules are created by the situation.

The second caveat is that the effects due to the rule application (firing) are probablistic. This means that executing a sentence under some state Q may lead to effect Υ_1 in 90 percent of the trials and Υ_2 in the other 10 percent of the trials. To represent this, the value P_Υ is maintained for each condition/sentence/effect triple. Expert systems and predicate calculus based planners usually account for only deterministic actions.

The third caveat is that the rules in the Organizer maintain a cause/effect model which describes the state evolution of objects due to actions by the Machine. This will facilitate goal planning as discussed later in this work.

The intuitive notion behind PLAY can be likened to an infant experiencing the world for the first time. By attempting to perform actions in the world, the infant determines what acts he can perform using his arms, legs, fingers, etc., and how hard it is to perform those acts. Also, the infant discovers how he can manipulate objects in his world, and how he affects the world through his actions.

Similarly, PLAY allows the Intelligent Machine to experience its abstracted world. It attempts to perform actions in its environment and receives feedback from the lower levels which analyze the difficulty of performing the tasks. This difficulty measure is stored in the rules, and is also applied to the complexity model which is used by higher reasoning when attempting new tasks. By interacting with the environment, the machine formulates new rules which reflect the changes in the world due to rule execution. Old rules can be modified when their effects on the world become more certain or less certain. Over time, the Machine builds up a robust store of these concepts.

It is now necessary to describe the PLAY algorithm in conceptual terms. Initially, the set of all rules which can be used in the current environmental state are gathered. This is called the active rule set, or R_{active}. If no rules match the current state, or a new rule generation has been selected, we proceed to formulating a new rule. The value p allows the Organizer to build new rules, and experiment with old rules during PLAY mode.

If a new rule must be generated, we select an actor, action, direct object and indirect object according to prespecified probability density functions. If the pdfs are uniform, the sentence is randomly generated. A non-uniform pdf allows a user to guide the Organizer during PLAY to discover certain types of rules which tend to contain a particular actor, action, etc. Therefore, a user can "supervise" the actions of the machine during PLAY, if desired.

The newly generated sentence is passed to the Coordination level and is executed. At the same time, the Organizer finds all the rules which have the same condition/sentence pair and calls this set R'_{active}. The determination of this set is necessary in order update the complexities, $\mathbf{H}$ and effect probabilities, $\mathbf{P}_{\Upsilon}$ of the rules. The set R'_{active} contains all the experienced effects of executing the sentence in the given environmental state.

Based on the feedback value, $\mathbf{H_c}(\mathbf{s})$, the Organizer determines whether the sentence was successfully executed. If it wasn't, the complexity values of all the active rules with the same sentence are updated. Updating all rules in R'_{active} reflects the fact that the complexity value describes the difficulty of executing a sentence in a particular state. Therefore, all rules which have the new sentence and the given state must have their complexities modified. The complexity model $\mathcal{B}_c$ is also trained with this information. If the newly formed sentence is the only member of this set (no other rules in R'_{active}), it is tossed away and the process begins again.

If the sentence can execute, the complexity values of the rules in R'_{active} are updated to the feedback complexity, and $\mathcal{B}_c$ is trained on this data. The new environmental state vector, $\mathbf{Q}'$ is observed. If the rule formed by combining the old state, sentence, and new state is not in the rule base, it is added. The probabilities for all members in R'_{active} are then modified to reflect the effect caused by applying the sentence in the old environment. The rule with the correct effect has its probability value $\mathbf{P}_{\Upsilon}$ increased, and all other rules in R'_{active} have their values decreased. The process of PLAY then begins again.

If the creation of a new rule is not called for, the process is somewhat similar. Of the rules in the active set, one is selected for execution based on the probability density function P_R. If P_R is uniform, each rule is selected with equal probability. However, the user can adjust P_R to guide or supervise the system to execute particular sentences and learn the effects of certain classes of sentences.

As described above, the sentence is passed to the Coordination level, evaluated, and in similar fashion, the rule complexities and probabilities are updated and the complexity model is trained. If a new environmental state is found, a new rule is formed and added to the rule store.

The PLAY operation should continue until the user decides the Machine has developed a rich enough model of its environment. It is important to realize that during PLAY, the Machine may be interacting with a simulator providing all the responses of the real world. This is one way to prevent the Machine from carrying out rules which have disastrous ends. Another method is through the adjustment of the rule selection probabilities, which allows supervision of the rules that the Organizer chooses to discover.

3.3.2 Advanced Mechanisms and Operation

Given a complete description of the basic mechanisms and the basic method of operation, it is possible to describe the advanced structures in the Organizer model. These advanced mechanisms facilitate the formation of plans by abstracting knowledge maintained in the basic mechanisms without destroying the original information. The advanced mechanisms in the Organizer are:

1. The generalizer.

2. The Boltzmann Machine for directed exploration.

The next sections describe the function of each unit and how it operates.

Symbolic Learning and The Generalizer

Many researchers have developed learning techniques for expanding and generalizing knowledge from rules in a rule base or from the examination of rule execution [And83, Car86, DeJ86]. Since the Organization level maintains a rule base of condition/sentence/effect rules, many of these symbolic learning systems can be overlayed on the Organizer in the same manner they can be overlayed on a top-down symbolic system such as STRIPS or ABSTRIPS. As an example, the Generalizer is presented as one type of symbolic algorithm which removes preconditions from a rule in the rule store to allow the rule to match more environmental conditions.

The PLAY algorithm is the basic mechanism used to explore the abstracted environment and formulate new rules based on the effects that a particular binary sentence Σ_i had on a given environmental state. Through this exploration, PLAY forms rules which are specific to a particular environmental instance Q. It is very likely that many of the states in Q are not affected by the execution of Σ_i. This implies that certain $q_k \in Q$ will not change state when a particular Σ_i is executed. Other fields in Q are highly likely to change state, namely those which are within the *locality of effect* as defined earlier. It would be helpful if only the relevant $q_k \in Q$ (those likely to change state) were represented in rules, while the irrelevant fields were ignored.

Ignoring the fields outside the locality of effect allows a specific rule formed during PLAY to become more general by allowing application of the rule to a host of abstracted states. The rule could effectively disregard object states which do not matter. For example, when sharpening a pencil, one is not concerned with a ball in the corner of the room. Similarly, when getting a glass of water, the state of the television is not highly relevent.

By increasing the generality of rules, one can also abstract knowledge into untested domains. For example, if it turns out that filling a glass with water can be done with the television on, the rule may be generalized to say that it is possible to fill a glass of water regardless of the state of the television. In this example, generalization has allowed the rule to apply to a state which has not yet been tested, namely when the television is off. This type of generalization works because the television is outside of the locality of effect.

The Generalizer works by removing preconditions of rules. This is done by creating a rule which contains a # (don't care) in a condition field which was previously occupied by a 1 or 0. The new rule can now match environmental states which have either a 1 or 0 for that field. The effect string is also modified to contain a # in the effect field which is at the same location in the effect string as the modified condition field in the condition string. This allows the 1 or 0 value which matches the # in the condition to be carried over to the effect.

For example, given the generalized rule:

1110#/1010101/1111#

The condition of this rule matches environmental states:

11100

and

11101

The effect string when the environmental state is 11100 would be

11110

The effect string when the environmental state is 11100 would be

11111

because the value assigned to the # field in the condition is passed through to the # of the effect.

Before introducing the generalizer algorithm, some notation is needed. For rule R_i, let Ξ_i^k be the substring of Ξ_i which contains all the states which observe object k. Similary, let Υ_i^l be the substring of Υ_i which contains all the states which observe object l. Let $\Xi_i^{\neg kl}$ and $\Upsilon_i^{\neg kl}$ be the substrings of Ξ_i, Υ_i respectively which do not contain any states which observe objects k or l. Also, Let $R_G = (\forall R_j \in \mathbf{R} : \# \in \Xi_j)$. This is the set of all generalized rules. Finally, let $R_S = \neg R_G$. This is the set of all specific rules.

Then the following algorithm allows for the generalization of rules:

A Generalization algorithm

Given a rule store **R**, a probability matching threshold θ_Υ and a complexity matching threshold θ_H:

1. Select a binary sentence, Σ_k.

2. Map Σ_k to the sentence s_k.

3. Let d_i be the direct object and i_i be the indirect object of s_i. Let ω_d be the object in Ω represented by d_i and let ω_i be the object in Ω represented by i_i. For shorthand, these can be called objects i and d.

4. Let $R_{match} = (\forall R_j \in R_S : \Sigma_j = \Sigma_k)$.

5. For each pair of rules R_a, R_b in R_{match} determine if the following is true:

   (a) $\Xi_b^d = \Xi_a^d, \Xi_b^i = \Xi_a^i$.

   (b) $\Upsilon_b^d = \Upsilon_a^d, \Upsilon_b^i = \Upsilon_a^i$.

   (c) $\Xi_b^{\neg di} = \Upsilon_b^{\neg di}$.

   (d) $\Xi_a^{\neg di} = \Upsilon_a^{\neg di}$.

   (e) $\| \mathbf{H_a} - \mathbf{H_b} \| \leq \theta_H$.

   (f) $\| \mathbf{P_\Upsilon^a} - \mathbf{P_\Upsilon^b} \| \leq \theta_\Upsilon$.

6. If true, do:

   (a) Create a new rule R_g, where $\Xi_g = \Xi_a$ except all fields $\Xi_g^{\neg di} = \#$, $\Sigma_g = \Sigma_a$, and $\Upsilon_g = \Upsilon_a$ except all fields $\Upsilon_g^{\neg di} = \#$.

   (b) Set $\mathbf{H_g} = \frac{\mathbf{H_a}+\mathbf{H_b}}{2}$.

   (c) Set $P_\Upsilon^g = \frac{\mathbf{P_\Upsilon^a}+\mathbf{P_\Upsilon^b}}{2}$.

   (d) If $R_g \in \mathbf{R}$ discard R_g.

   (e) Determine the total support for R_g and check the validity of the generalization (as described below).

   (f) If R_g is VALID, add R_g to **R**.

7. Go to 1.

Default hierarchies and valid generalizations

Together, the general rules (those with # as fields) and the specific rules (those without # as fields) form default rule hierarchies. As discussed in [HHNT86], a default hierarchy is a set of rules which contain general rules of thumb, specific instantiations of these general rules, and exceptions to the general rule. By using an algorithm such as the one above for generalization, general rules can be formed and applied to domains where specific rules do not exist, while specific rules are available to represent instantiations of, and exceptions to the general rule. The main difference between Holland's default hierarchy and this version of the Organizer's is that the former is built from general rules to specific instances, while the latter is constructed from specific instances to general rules. Using the generalization algorithm above, the Organizer's hierarchy is limited to two levels: 1) Generalized rules for fields outside the locality of effect and 2) specific rules.

For the default hierarchy to remain effective, bad or conflicting generalizations must be eliminated and existing ones must be updated to reflect newly acquired information. To do this, we must find the set of rules which support the generalization and those which conflict with it. The following scheme allows for the removal of bad or conflicting generalizations by checking the validity of a generalized rule R_g.

Given a rule store **R**, a generalized rule R_g, a generalization acceptance threshold $0 \leq \theta_G \leq 1$, a probability matching threshold θ_Υ and a complexity matching threshold θ_H, and a function INSTANTIATE(R_s, R_g) defined as follows:

1. For each field $\xi_s^k \in \Xi_s$ and $\xi_g^k \in \Xi_g$, if $\xi_g^k = \#$ then set $\xi_g^k = \xi_s^k$, $(0 \leq k \leq \mathbf{M} - 1)$.
2. For each field $v_g^k \in \Upsilon_g$, if $v_g^k = \#$ then set $v_g^k = \xi_g^k$, $(0 \leq k \leq \mathbf{M} - 1)$.

The following is a test for validity of a generalization:

1. Let $R_{spec} = (\forall R_k \in R_S : \Sigma_k = \Sigma_g \text{ AND } \Delta(\Xi_k, \Xi_g) = 0)$. This is all the specific rules in **R** which can be instantiated from the generalization R_g.
2. Let $TOTAL_{support}$ equal the number of specific rules which support R_g. Initialize $TOTAL_{support}$ to 0.
3. For each $R_s \in R_{spec}$, INSTANTIATE(R_s, R_g). If:
   (a) $\Delta(\Upsilon_s, \Upsilon_g) = 0$, and
   (b) $\|\mathbf{H_s} - \mathbf{H_g}\| \leq \theta_H$, and
   (c) $\|\mathbf{P}_\Upsilon^{\mathbf{s}} - \mathbf{P}_\Upsilon^{\mathbf{g}}\| \leq \theta_\Upsilon$.
4. OR:
   (a) $\Delta(\Upsilon_s^d, \Upsilon_g^d) \neq 0$ or $\Delta(\Upsilon_s^i, \Upsilon_g^i) \neq 0$

5. then $TOTAL_{support} = TOTAL_{support} + 1$.

6. If $\frac{TOTAL_{support}}{||R_{spec}||} \geq \theta_G$ then the R_g is VALID.

Discussion on generalization

Generalization allows a wide range of rules to be summarized in a generalized rule. In a generalized rule, some fields which would normally be assigned 0 or 1 are replaced by #, which is the *don't care* symbol. Any field in the condition containing a # will match a 1 or a 0 of the current state vector. Thus, a rule containing *n don't care's* in the condition can match 2^n specific rules. The # symbol also allows knowledge abstraction to new domains as mentioned previously.

One type of generalization is accomplished by the algorithm above. This method generalizes across object states which are outside the locality of effect. Other types of generalization should be researched and developed which allow for generalization of actions for affected objects, generalization of effects for particular actions, etc. Also, more research must be done to build a more robust default hierarchy, each level of which handles small exceptions to rules at the previous level.

The generalization process can run in the background while PLAY or PLAN is being executed, or it can run while the Coordination and Execution level are executing a particular plan since it can search the rule store without interfering with these processes. The background process will create new generalizations and remove bad ones which will immediately take effect in PLAY mode, or wait until plan completion during PLAN mode.

The algorithm presented above can be summarized as follows:

For a particular sentence, find all the rules which contain the mapped sentence. This is called R_{match}, or the matched rule set. Obviously, since the sentences are identical, all rules in the matched set have the same direct and indirect objects. For each pair of rules, R_a and R_b, in the matched set, if the fields relevant to the direct object in the condition of R_a are identical to the fields in R_b and the same holds for the indirect object fields it means that the rules within the locality of effect are identical. Next, if the fields which are not relevant to the direct or indirect objects in the condition of R_a are identical to those fields in the effect of R_a, it means that the rule does not change states outside of the locality of effect. If the same holds for R_b, and the complexities and probabilities of effect are similar across the two rules, then the rules can be generalized. Generalization involves the replacement of all fields outside the direct and indirect object fields with #. This indicates that the field is now *don't care*, and can match either a 1 or a 0.

The function INSTANTIATE generates a specific rule from a generalized one. Given R_s, the specific rule and R_g, a general one, it sets all the # fields in the condition of R_g to be identical to those in R_s. The # fields in the effect part of a generalized rule are direct mappings of the respective fields in the condition part. This indicates that the rule does not effect those states, i.e. they remain the same.

Therefore, the # fields in R_g are assigned the respective values from the condition of R_g.

The validity of a generalization is examined by finding all the specific rules which match the condition and sentence of the generalization, and counting the number which support the effect, complexity and probability of effect denoted in the general rule. Rules which don't have the same effect on the direct or indirect objects don't negatively influence the support, because generalization conflicts caused by the generalization algorithm presented above are due in large to cases where locality of effect does not hold. By computing the ratio of generalization support to total number of matches, the validity of the generalization is assessed.

Example of generalization

We can use an extension of an examples presented earlier to demonstrate the generalization process.

Let:

$\Omega = (\omega_0, \omega_1, \omega_2, \omega_3, \omega_4) = (bottle,\ table,\ cabinet,\ wrench,\ arm)$.

$Q_0 = (q_0^0, q_0^1, q_0^2, q_0^3, q_0^4, q_0^5)$.

$q_0^0 = 1$ if *bottle* is *full*, 0 otherwise.

$q_0^1 = 1$ if *bottle* is *halffull*, 0 otherwise.

$q_0^2 = 1$ if *bottle* is *empty*, 0 otherwise.

$q_0^3 = 1$ if *bottle* is *on − table*, 0 otherwise.

$q_0^4 = 1$ if *bottle* is *in − cabinet*, 0 otherwise.

$q_0^5 = 1$ if *bottle* is *carried − by − arm*, 0 otherwise.

$Q_3 = (q_3^0, q_3^1, q_3^2, q_3^3, q_3^4)$.

$q_3^0 = 1$ if *wrench* is *open*, 0 otherwise.

$q_3^1 = 1$ if *wrench* is *closed*, 0 otherwise.

$q_3^2 = 1$ if *wrench* is *on − table*, 0 otherwise.

$q_3^3 = 1$ if *wrench* is *in − cabinet*, 0 otherwise.

$q_3^4 = 1$ if *bottle* is *carried − by − arm*, 0 otherwise.

$\mathbf{Q} = (q_0^0, q_0^1, q_0^2, q_0^3, q_0^4, q_0^5, q_3^0, q_3^1, q_3^2, q_3^3, q_3^4)$.

Let $\mathbf{A} = (a_0) = (ARM)$.

Let $\mathbf{V} = (v_0,\ v_1,\ v_2) = (MOVE,\ GRASP,\ RELEASE)$.

Let $\mathbf{D} = \mathbf{I} = \mathbf{\Omega}$.

Assume we have two rules present in the rule store which have been created through PLAY. The first rule is:

$$01010010010/10101000000001/01000110010$$

denoting:

Condition: (*bottle − halffull, bottle − on − table, wrench − open, wrench − in − cabinet*).

Rule: (*arm grasp bottle arm*)

Effect: ($bottle - halffull$, $bottle - carried - by - arm$, $wrench - open$, $wrench - in - cabinet$).

and the second rule is:

01010001100/10101000000001/01000101100

denoting:

Condition: ($bottle - halffull$,$bottle - on - table$, $wrench - closed$, $wrench - on - table$).

Rule: (*arm grasp bottle arm*)

Effect: ($bottle - halffull$, $bottle - carried - by - arm$, $wrench - closed$, $wrench - on - table$).

Assuming similar probabilities and complexities (within threshold margin), we can generalize to create the general rule:

010100####0/10101000000001/010001####0

which replaces the fields relevant only to the wrench, table and cabinet by ***don't care***. It is important to note that since the *arm* was the indirect object in the sentence, the field $wrench - carried - by - arm$ could not be replaced. Intuitively, it can be seen how the generalization allows the rule to apply to 4 specific cases where only 2 specfic rules were needed to generate it.

A Boltzmann Machine for Directed Exploration and Learning

The PLAY algorithm provides an effective means for developing rules for the rules store and testing these rules to determine the probability of effect and complexity values. The PLAY algorithm can be guided by the user to examine particular rules, but it is not goal directed. It is important that the Organizer contain a mechanism which allows for goal directed exploration of its actions in a manner which also minimizes the cost of exploring. The goal-directed behavior is the topic of this section.

An introduction to the architecture designed to achieve this functionality is presented in [SM88, MS89]. A more formal, expanded version is presented here.

The function of this unit is to facilitate goal-directed exploration of the environment and determination of subgoals during planning. Since goal directed behavior implies an attempt to pursue a method for modifying an objects state in a given way, a goal-directed Boltzmann Machine, denoted $\mathcal{B}_g$, is designed to model the change of state of objects due to the execution of particular sentences.

A Boltzmann Machine is a neural network that provides associative recall by minimizing the Energy, which is a measure of correlation between the asserted nodes in the network. Placing the goal-directed Bolzmann Machine within the framework of the Intelligent Machine, we can define the Energy of the Machine as Knowledge about a particular state (K). Since the minimization of Energy yields the correct associative response, this maximizes the Knowledge about the state. Again, minimizing Energy of the state of the Boltzmann machine maximizes the Knowledge about that state.

Training the goal-directed Boltzmann Machine modifies the weights of the network and alters the Energy function. Therefore, training updates the Knowledge in the Machine over time, and can be defined as the Rate of Machine Knowledge (R).

The $\mathcal{B}_g$ network is trained to search for a binary sentence which produces the given object state changes by maximizing the Knowledge about the state of the network. Combined with information from the complexity model $\mathcal{B}_c$, this unit will produce low-complexity sentences which have a high probability of changing given object states. These sentences will be used to form steps in subtasks or *skills*. Skills, which are called schema in Artificial Intelligence, are sequences of sentences which achieve a subgoal. Skills allow reduction in planning search time by collapsing many search steps into a single one.

Consider the earlier example of a mobile robot which must use a wrench to shut off a pipe. The goal of this plan is to shut off the pipe, while a subgoal might be "get wrench from open tool-chest". It is the responsibility of this system to create a set of sentences which can be combined to achieve this subgoal. This set of sentence is a skill.

To achieve these capabilities the goal-directed Boltzmann Machine is provided with a desired set of object state changes on its input nodes. The Machine then searches the sentence nodes to find the proper combination of actor, action, direct object and indirect object which maximizes the Knowledge about the state. Using this value, K, the analytic formulation allows the computation of the Probability that the computed binary sentence is correct, and the Uncertainty associated with the sentence given the desired state changes. Sentences with low uncertainty form subtasks of the subgoal plan. A subgoal plan is developed by using a graph search technique with large predictive value placed on use of the discovered subtasks.

The goal-directed Boltzmann Machine is defined as follows:

The $\mathcal{B}_g$ network is divided into two parts, $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$. The $\mathcal{B}_g^+$ network models object states which switch from 0 to 1 in the state vector Q. The $\mathcal{B}_g^-$ network models states which switch from 1 to 0.

For example, assume we wish to change from state Q $=$ 10010 to Q $=$ 00011. We see that q_0 changes from 1 to 0 and q_4 changes from 0 to 1. This information about q_0 would be input to $\mathcal{B}_g^-$, while the information about q_4 would be input to $\mathcal{B}_g^+$. It is the responsibility of $\mathcal{B}_g^+$ to search for the binary sentence which maximizes the Knowledge (i.e. minimizes the Energy) of the network given 00001 as input and the responsibility of $\mathcal{B}_g^-$ to search for the binary sentence which maximizes K given 10000 as input.

The $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$ networks are each identical to the $\mathcal{B}_c$ network in terms of node levels and connections, i.e.:

Let $\mathcal{B}_{\mathbf{g}}^+ = (\mathcal{L}, \mathcal{W})$ be a connectionist network. Let node levels $\mathcal{L} = (L_Q, L_{QQ}, L_A, L_{AV}, L_V, L_{VD}, L_D, L_{VI}, L_I)$. Similarly, let $\mathcal{B}_{\mathbf{g}}^- = (\mathcal{L}, \mathcal{W})$ be a connectionist network.

Let node levels $\mathcal{L} = (L_Q, L_{QQ}, L_A, L_{AV}, L_V, L_{VD}, L_D, L_{VI}, L_I)$. The node level description, $\mathcal{L}$ is identical in $\mathcal{B}_c$, $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$.

Because $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$ are Boltzmann machines, we use an Energy function to describe the individual output of these two networks. Energy output values must be greater than or equal to 0.0 because negative Energy does not exist in such a system. Therefore, we will restrict our weight values to:

$$0.0 \leq w_{ij} \leq 1.0$$

The following rules hold for $\mathcal{W}$ in $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$:

1. Some weights are fixed at 0.0. Other weights can vary between 0.0 and 1.0.
2. If w_{ij} is fixed at 0.0, there is no connection between nodes n_i and n_j. Else, a connection exists between nodes n_i and n_j.

Similar to $\mathcal{B}_c$, we denote the **Knowledge about a particular state** n **of** $\mathcal{B}_g^+$ as $K(n)$ where n represent the state of the nodes. This values is defined as:

$$K_g^+ = K(\mathcal{B}_g^+(n)) = \sum_{i=0}^{\|\mathcal{L}\|}\sum_{j=i}^{\|\mathcal{L}\|} w_{ij}n_in_j$$

.

Also,

$$K_g^- = K(\mathcal{B}_g^-(n)) = \sum_{i=0}^{\|\mathcal{L}\|}\sum_{j=i}^{\|\mathcal{L}\|} w_{ij}n_in_j$$

.

Minimizing K_g^+ or K_g^- results in maximizing $K(n)$.

Knowledge in this system relates the amount of correlation between pairwise asserted node combinations. The higher the weight w_{ij}, the less correlated the two nodes n_i, n_j. The lower the weight, the higher the correlation. Correlation describes the assessment over time of the chance that two nodes are asserted simulataneously. As discussed previously, this pairwise correlation is important in determining which binary sentence nodes should be asserted in order to find the best match for asserted state change nodes.

Given a desired change of state, the purpose of each Boltzmann machine is to output high $K(n)$ (low Energy) values for sentences which have a high probability of accomplishing that state change. To produces the maximum $K(n)$ output, a search technique must be employed to examine the "goodness" of binary sentences, and select new sentences to test based on some performance criteria, i.e.:

1. Let δQ represent the state vector of states which change from 0 to 1. Any field $q_i \in \delta$Q is set to 1 if it must change from 0 to 1, else it is set to 0.

2. For each field q_i in δQ, if $q_i = 1$ then set $n_Q^i = 1$ in $\mathcal{B}_g^+$.
3. Search the node levels L_A, L_V, L_D, L_I to maximize $K(n)$ of $\mathcal{B}_g^+$.

The same method holds for $\mathcal{B}_g^-$ except we use δQ to represented object states which change from 1 to 0.

Search techniques

Three random search techniques are compared here which may be used to find the minimum Energy in a Boltzmann Machine.

A genetic algorithm search technique

A technique which minimizes a system cost function is the Genetic Algorithm [Hol75]. In contrast to other random search techniques, the Genetic Algorithm (GA) maintains a population of points in the space while searching for the optimum.

Here we present a modified GA which will converge in probability to the minimum cost. The standard GA has been changed by inserting spacer steps of an algorithm which is known to converge in probability, Expanding Subinterval Random Search.

Spacer steps are defined as follows: Suppose B is an algorithm which together with a descent function Z and solution set T converges in probability. We can define an algorithm C by $C(x) = \{y : Z(y) \leq Z(x)\}$. In other words, C applied to x can give any point so long as it does not increase the value of Z, the current cost. B represents the spacer step, and the complex process between spacer steps is C. Thus, the overall process amounts to repeated applications of the composite algorithm CB. CB will converge in probability if B is repeated infinitely often and C does not increase the value of the current cost [Lue84].

We introduce the concept of *immigration* to imbed ESRS into GA. Infinitely often, we insert a randomly generated point into the GA search which forms the spacer step. The frequency of insertion is called the *immigration rate*. By changing the immigration rate, the algorithm adjusts its focus from global to local searches. This rate may be fixed dependent on the complexity of the search space, or may vary while the search is in progress. A high immigration rate will force random search. A low rate will cause the GA. Parallels can be drawn to Simulated Annealing which starts as a near random search, and eventually becomes gradient descent. For the modified GA, the immigration rate is analogous to thermal energy in Simulated Annealing. The modified algorithm described in detail below converges in probability to the minimum cost.

In general, for Holland's Genetic Algorithm, each point in the space is represented by a binary string and has an associated cost dictated by the system cost function for that point. Since the makeup of the population is changed each iteration to emphasize members (points) which minimize the cost function, a near-uniform population will develop corresponding to a local minima in the cost function.

The following notation is used:
P = population of members (points)
P' = new population of members
$|P|$ = number of members in P
P_k = kth member of the population P
$P_k(m)$ = mth bit of P_k
J_k = cost of P_k
S_k = probability of member k being selected from current population
J_{max} = max cost of an possible member in P
n = length of P_k in bits

Each iteration of the standard GA search algorithm proceeds as follows:

```
Repeat until ($P_k \in P$ and $P_k$ has minimum cost)
{ 1. Compute $J_k$, $\forall P_k \in P$.
  2. Let $J'_k = J_{max} - J_k$, $\forall k$. Compute $S_k = J'_k/(\sum_i J'_k)$ , $\forall k$.
  3. Repeat until $|P'| = |P|$
    { 3a. Randomly select $P_j, P_k$ from $P$ based on $S_j, S_k$.
      3b. Randomly generate an index $i$ between $1..n$.
       Exchange the right string halves of $P_j, P_k$
       (i.e. $P'_j(i..n) = P_k(i..n)$ and $P'_k(i..n) = P_j(i..n)$).
       This is called "crossover" or "mating".
      3c. Place $P'_j, P'_k$ in $P'$. Return $P_j, P_k$ to $P$.
  4. Set $P = P'$.
```

In an attempt to prevent population convergence on a local minima (premature convergence), a *mutation* operator is added to the system. With a new generation of the population, each bit of every member has a small probability of inverting. The inversion adds diversity to the population and promotes search in previously unexplored regions of the space in an attempt to find the global cost minimum.

Particular aspects of this algorithm make it a powerful search tool. The *crossover* mechanism forces search on an n-dimensional hypercube by discovering and promoting particular substrings (called *building blocks*) which perform well. These building blocks combine to discover the topology of the search space, which may not be known initially. Since the algorithm uses a population of points, many planes of the hypercube can be searched at once, leading to implicit parallelism. Further, since members within a population are independent, a new population may be formed by mating in parallel. Steps 3a-3c can be blocked together and generate two new members in parallel with other mating blocks. These features as well as others are described in depth in [Gol89]. Applications of this algorithm are presented in [DJ75, GGRG85, DC87].

Heuristic algorithms have been developed within GA to avoid convergence at local minima [Mau84, SG87]. The "SIGH" system [Ack87] uses active and passive subpopulations to escape local minima. When particular members of the population are performing poorly, they become passive until the active subpopulation converges. If this convergence is premature, the passive members are activated, bringing diversity and new structure to the search.

Unfortunately, many of the heuristically driven GA searches perform well for a small set of functions, and prematurely converge for functions outside that set. However, it can be shown that under certain conditions, the GA will converge in probability to the global minimum of the cost function.

Theorem 3.2: The Genetic Algorithm (GA) converges in probability to the global minimum of a cost function Z if:

1. Instead of (or in addition to) the mutation operator, an *immigration* operator is used. Introduce a member P_i' generated randomly from a uniform density function to population P' every M populations for some integer $M > 0$.

2. If $P_k \epsilon P$ and $\forall P_i \epsilon P, J_k \leq J_i$ then $P_k \epsilon P'$. In other words, the best performing string in the current population is placed in the next population.

Proof:

1. Let us define a function G which at iteration i generates P_i' which is a randomly selected state vector from a prespecified i.i.d. density function. Call G the *immigration function.*

2. Let B be a function which generates a state vector X_{i+1} at iteration $i+1$ based on:

$$X_{i+1} = \begin{cases} P_i' & \text{if } Z(P_i') - Z(X_i) \leq 2\mu \\ X_i & \text{if } Z(P_i') - Z(X_i) > 2\mu \end{cases}$$

   where $Z(X)$ is a descent function for the state vector X, and μ is 0 since the cost function is deterministic.

3. The function B is called Expanding Subinterval random search and is known to converge in probability to the minimum of $Z(X)$.

4. Let C be the Genetic Algorithm described above with the modification that if $P_k \epsilon P$ and $\forall P_i \epsilon P, J_k \leq J_i$ then $P_k \epsilon P'$.

5. In C, let the evaluation function $J_k = Z(P_k)$.

6. At any iteration of C during a particular iteration i of B, let $X_i = (P_k \epsilon P : \forall P_i \epsilon P, Z(P_k) \leq Z(P_i))$

7. Then C is an algorithm such that $C(x) = \{y : Z(y) \leq Z(x)\}$

8. Let us form a process CB from repeated iterations of C and repeated iterations of B in any order with the constraint that B be repeated infinitely often.

9. Then the algorithm CB converges in probability to the minimum value of Z.

This proof inbeds ESRS into the GA, where ESRS is algorithm B as described by [Lue84] and stated above. It insures $C(x) = \{y : Z(y) \leq Z(x)\}$ where C is the GA algorithm. Therefore, CB, the modified GA, converges in probability to the cost minimum.

As one can see, these necessary conditions do not bind the algorithm severely. The immigration rate (immigrations/population), 1/M, is related to the mutation rate (mutations/bit) as follows:

$$1/M = (\text{mutations/bit}) * (\text{members/population})$$

In fact, the immigration of new members may be probabilistic, with probability 1/M.

Simulated annealing

One random search technique commonly used to find the global minimum cost in a Boltzmann Machine is Simulated Annealing. This technique simulates the annealing process of metal by probabilistically allowing uphill steps in a state–dependent cost function while finding the global cost minimum, or ground state. The algorithm allows control of the search randomness by a user specified parameter, T. In true metal annealing, this cost function is the Energy of the system, E, and T is the annealing temperature [KJV83].

Given is a small random change in the system state $X_i = \{x_1, x_2, \ldots, x_n\}$ to X_i' and the resulting Energy change, δE, if $\delta E \leq 0$, the change is accepted. If $\delta E > 0$, the probability the new state is accepted is:

$$p(X_{i+1} = X_i') = e^{-\delta E/K_B T}$$

where K_B is the Boltzmann Constant and T is a user set parameter. By reducing T along a schedule, called the annealing schedule, the system should settle into a near–ground state as T approaches 0.

Another method for simulated annealing is discussed in [HS86]. Using this method, if the Energy change between X_i and X_i' is δE, then regardless of the previous state, accept state X_i' with probability:

$$p(X_{i+1} = X_i') = \frac{1}{1 + e^{-\delta E/T}}$$

Since an Boltzmann Machine consists of a set of binary states, it should be noted that in both of the above methods, X_i' is hamming distance 1 from X_i.

The process of simulated annealing escapes local minima through its probabilistic random search, and probabilistically convergences to the global cost minimum under certain conditions [GG84]. The next technique, Expanding Subinterval Random Search, probabilistically guarantees convergence within a δ neighborhood to the global minimum of a specified cost function.

Expanding Subinterval Random Search

A third technique for finding the global minimum value for a cost function for a dynamic system is Expanding Subinterval Random Search as described in [Sar77]. Using Energy as the cost function and given a state X_i, one may define the following random search algorithm for an appropriately selected μ:

$$X_{i+1} = \begin{cases} X_i' & \text{if } E(X_i') - E(X_i) \leq 2\mu \\ X_i & \text{if } E(X_i') - E(X_i) > 2\mu \end{cases}$$

where $E(Y)$ is the Energy induced by state $Y = (y_1, y_2, \ldots, y_n)$ and X_i' is a randomly selected state vector generated from a prespecified independent and identically distributed density function.

It is shown that:

$$\lim_{n \to \infty} \textit{Prob}\ [E(X_n) - E^*_{\min} < \delta] = 1$$

where $E^*_{\min}$ is the global minimum Energy of the network. The existence of $E^*_{\min}$ is proven in the cited work.

This method can be used on–line to find the global minimum Energy in a Boltzmann Machine.

Experimental Results

A net was created which recognized strings of 15 bit binary numbers. This was done by creating a network of 15 nodes, each connected to every other. The net was formulated using the standard Energy methods found in [HS86].

The state of nodes 4 and 6 (000010100000000) were held at 1, which corresponded to a fixed input to the network. By changing the values of the other nodes in the network, the minimum Energy of the network could be found. For this purpose, search techniques were invoked to find the minimum Energy by altering the node values. The value of the nodes when the network was at minimum Energy formed an ordered binary string which was the correct associative recall of the network for the given input.

For the given input, the net had three Energy minima, corresponding to states (001010100100100, 110110110001101, 001111101100010) which were associative recall strings. The respective Energy for these three states were (0.8, 0.6, 1.0). Each simulation technique attempts to find the global Energy minimum of the net, which was 0.6, and corresponds to the correct associative response to the input. The cases presented here show best and worst performance of each technique over 10 trials. Other case which varied the depth and width of the Energy wells are presented in [SM88]. For this experiment, the wells were narrow.

The Modified Genetic Algorithm was performed with the added convergence techniques intact. The population was set at 20 members. Each member was 15 bits long, so the number of bits in each population was 300. The *immigration rate* was set to 0.5 which corresponds to a mutation rate of 0.025.

Simulated Annealing was performed and the system was cooled in accordance with:

$$\frac{T_1(t)}{T_0} = \frac{1}{\log(10+t)}$$

where $T_1(t)$ = temperature at time t
T_0 = initial temperature.

The net state changed in Hamming distance 1 increments.

Expanding Subinterval Random Search was slightly modified to reinforce the probabilistic selection of node states which reduced the Energy in the net. The probability of a node being active as initially 0.5. When the Energy was reduced during search, the probability of the node being reactivated became

$$P(x_i = 1) = P(x_i = 1) + [1.0 - P(x_i = 1)] * 0.1$$

if the node was active, or

$$P(x_i = 1) = P(x_i = 1) - P(x_i = 1) * 0.1$$

if the node was inactive.

Figures 3.6a - 3.6f present the best and worst performance of each algorithm over 10 trials. Modified GA found the minimum Energy string between the 20th and 180th population. Since there were 20 strings per population, this indicates that between 400 and 3600 points had to be generated. The best performance by Simulated Annealing required over 5500 iterations. The worst performance did not converge in 12000 iterations (the most attempted). As a guideline, the best performance of the random search ESRS was slightly over 2000 iterations. The worst performance did not converge in 12000 iterations. The results of these limited experiments force a closer examination of the Modified Genetic Algorithm as a search technique for minimizing the Energy in a Boltzmann machine.

Discussion on search techniques

The techniques discussed and simulated above are all valid for application to the $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$ networks. With the state change vector δQ fixed on the nodes, the L_A, L_V, L_D, and L_I levels can be searched to find the maximum $K(n)$. However, a few modifications to the search algorithms must be made.

1. Valid search strings must include exactly one asserted node in L_A, one asserted node in L_V, zero or one asserted nodes in L_D, and zero or one asserted nodes in L_I.

2. Along with a particular string asserted on the node levels, the respective nodes on levels L_{QQ}, L_{AV}, L_{VD}, L_{VI} must be asserted.

The minimum Energy search finds the binary sentence which has the highest probability of achieving the desired state changes.

Item one of the above list yields a high rate of non-allowable strings compared to allowable strings. Thus, the search techniques above may be inefficient at this task. Research must be done to investigate and discover a set of efficient search techniques which can find the minimum Energy in Boltzmann Machines of this nature.

Combined search of $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$.

The minimization techniques presented above independently find a binary sentence which maximizes $K(n)$ (minimizes the Energy) for $\mathcal{B}_g^+$ and another binary sentence which maximizes $K(n)$ for $\mathcal{B}_g^-$. This technique may lead to the discovery of two different binary sentences. To eliminate this possibility, let us define the $K_g = \eta K_g^+ + (1-\eta)K_g^-$, where $0.0 \leq \eta \leq 1.0$. Further, let us force each network to assert the same binary sentence at each search step:

1. Generate a binary sentence Σ_i.

2. Assert Σ_i on nodes in both $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$.

3. Assert the pairwise node levels.

4. Compute K_g.

5. Go to 1.

This will force the system to find a sentence which accomplishes both sets of state changes simulataneously and will maximize $K(n)$ by minimizing K_g which is the combined state change performance.

How good is the found sentence?

Given a binary sentence, Σ_i found by maximizing $K(n)$ for $\mathcal{B}_g$, Σ_i is the best available sentence which may achieve the desired state changes. The questions arise: How good is Σ_i at actually changing the desired states? How likely is it to change other states instead?

One method for examining the "goodness" of a sentence in acheiving the desired results is by looking at the Entropy of $\mathcal{B}_g$ for the given sentence. If the system has low

Entropy for the given sentence, the sentence is likely to affect the states as desired. If the system has high Entropy for Σ_i, the sentence may not affect the states in the desired way, and may actually affect other states.

The Entropy value can be derived from the Knowledge of the states of $\mathcal{B}_g^+$ and $\mathcal{B}_g^-$. We know that:

$$K(\mathcal{B}_g^+(n)) = \sum_{i=0}^{\|\mathcal{L}\|} \sum_{j=i}^{\|\mathcal{L}\|} w_{ij} n_i n_j$$

Using the analytical model of the Intelligent Machine, we can find the **Probability that the System is in State** n:

$$P(\mathcal{B}_g^+(n)) = e^{-\rho - K(\mathcal{B}_g^+(n))}$$

where ρ is a probability normalizing factor, and we can define:

$$H(\mathcal{B}_g^+(n)) = -\sum_n P(\mathcal{B}_g^+(n)) ln\{P(\mathcal{B}_g^+(n))\}$$

which is the **Uncertainty that the System is in State** n.

Similar definitions are used for $\mathcal{B}_g^-$. In the above equations, n represents the state of the nodes in the network.

The value H is an Entropy measure which gives the uncertainty of knowledge given a particular binary sentence Σ_i. This Entropy value reflects the uncertainty that the Intelligent Machine changes only the desired object states given the Σ_i.

To compute the uncertainty value $H(\mathcal{B}_g^+(n))$ given Σ_i for network $\mathcal{B}_g^+$:

1. For each field σ_i in Σ_i where $(0 \leq i \leq \|\mathbf{A}\| - 1)$, if $\sigma_i = 1$, then set $n_A^i = 1$.
2. For each field σ_i in Σ_i where $(\|\mathbf{A}\| \leq i \leq \|\mathbf{V}\| - 1)$, if $\sigma_i = 1$, then set $n_V^i = 1$.
3. For each field σ_i in Σ_i where $(\|\mathbf{V}\| \leq i \leq \|\mathbf{D}\| - 1)$, if $\sigma_i = 1$, then set $n_D^i = 1$.
4. For each field σ_i in Σ_i where $(\|\mathbf{D}\| \leq i \leq \mathbf{N} - 1)$, if $\sigma_i = 1$, then set $n_I^i = 1$.
5. For each pair of nodes n_Q^i and n_Q^j $(i < j)$, if both nodes equal 1 then $n_{QQ}^{ij} = 1$.
6. For each pair of nodes n_A^i and n_V^j, if both nodes equal 1 then $n_{AV}^{ij} = 1$.
7. For each pair of nodes n_V^i and n_D^j, if both nodes equal 1 then $n_{VD}^{ij} = 1$.
8. For each pair of nodes n_V^i and n_I^j, if both nodes equal 1 then $n_{VI}^{ij} = 1$.
9. Set all other nodes to 0.
10. Set ρ to 0.
11. For each possible state vector $\delta\mathbf{Q}$:

(a) For each field q_i in δQ, if $q_i = 1$ then set $n_Q^i = 1$, else set n_Q^i to 0.

(b) Compute K_g^+.

(c) Compute the probability value $P(\mathcal{B}_g^+(n))$.

12. Sum the probability values computed for each state δQ and find the necessary probability normalizing factor ρ so the sum equals 1.

13. Normalize the probabilities by $e^{-\rho}$.

14. Compute $H(\mathcal{B}_g^+(n))$.

A similar algorithm can be used for $\mathcal{B}_g^-$.

This algorithm is of order $O(2^M)$ where M is the length of the state vector Q. To reduce the computational complexity of this algorithm, one can employ the locality of effect premise. Under this assumption, the only states which need be considered are those containing the direct or indirect object of Σ_i. If ω_d is the direct object and ω_i is the indirect object the order of the algorithm becomes $O(2^{m(d)+m(i)})$, $(m(d)+m(i) \ll M)$.

Goal-directed exploration

Goal-directed exploration is the process of changing particular object states in the state vector Q such that the new state vector Q′ is closer (in a Hamming distance sense) to a particular goal vector Q^*. That is $\Delta(Q,Q^*) > \Delta(Q',Q^*)$. Given a Boltzmann machine as described above, how can it be employed for goal-directed exploration?

Provided with a given state change vector δQ it is desirable for the Boltzmann machine $\mathcal{B}_g$ to maximize $K(n)$ in order to produce a binary sentence which accomplishes the necessary change of state. However, it is possible that the binary sentence, although very likely to accomplish the task in some environmental states, cannot perform in the current environmental state Q. How can this be accounted for?

One method for accomplishing this type of exploration is by searching the networks $\mathcal{B}_g^+$, $\mathcal{B}_g^+$, and $\mathcal{B}_c$ simultaneously. Let us assign a measure $K_d = \phi\lambda H_n(s) + (1-\phi)K_g$, where $0.0 \leq \phi \leq 1.0$. K_d represents the combined values of the outputs of the complexity and goal-directed networks. ϕ allows the system to tradeoff between the complexity and goal-directed outputs, while λ transforms the complexity output of the complexity model to a scale applicable to the goal-directed networks.

Given a maximum Entropy threshold θ_d which is the maximum allowable Entropy of a goal-directed sentence, the search for a low complexity sentence which is likely to produce the desired state changes can be described by:

1. Let δQ represent the state vector of states which change from 0 to 1. Any field $q_i \in \delta$Q is set to 1 if it must change from 0 to 1, else it is set to 0.

2. For each field q_i in $\delta\mathbf{Q}$, if $q_i = 1$ then set $n_Q^i = 1$ in $\mathcal{B}_g^+$.
3. Let $\delta\mathbf{Q}$ represent the state vector of states which change from 1 to 0. Any field $q_i \epsilon\ \delta\mathbf{Q}$ is set to 1 if it must change from 1 to 0, else it is set to 0.
4. For each field q_i in $\delta\mathbf{Q}$, if $q_i = 1$ then set $n_Q^i = 1$ in $\mathcal{B}_g^-$.
5. For each field q_i in $\mathbf{Q}$, if $q_i = 1$ then set $n_Q^i = 1$ in $\mathcal{B}_c$.
6. Generate a binary sentence Σ_i.
7. Assert Σ_i on nodes in the three networks.
8. Assert the pairwise node levels in the three networks.
9. Compute K_d.
10. Until minimum K_d is found, go to 6.
11. Compute $H(\mathcal{B}_g(n))$.
12. If $H(\mathcal{B}_g(n)) \leq \theta_d$ then execute s_i.

This algorithm searches for the binary string which minimizes the combined Energy value of the three networks, finds the Entropy of the binary string in obtaining the desired state changes, and sends the string to the Coordination level if it is satisfactory.

Task Decomposition in Planning

The main function of the Organizer is the development of a plan **P** composed of a set of sentences $\mathbf{S}^*$ which accomplishes a defined goal $\mathbf{Q}^*$ from an initial state $\mathbf{Q}_0$. The networks $\mathcal{B}_g^+$ and $\mathcal{B}_g^+$ can be used to extract binary sentences which have a high probability of changing particular object states. Given $\mathbf{Q}^*$ and $\mathbf{Q}_0$, the object states which must change can be easily determined. Using different combinations of this state change vector, the goal-directed networks can develop a set of low Entropy binary sentences to accomplish subsets of state changes, or subtasks.

After a set of subtasks is found, a graph search algorithm such as the one presented in the next section can be used to develop skills which achieve task subgoals by sequencing the subtasks and adding additional rules if necessary. These skills will be placed in the rule store and can be extracted by the planner to form a task plan. This is one of the research directions proposed in this report.

A Planner can use these skills to reduce its search space when planning for a goal, since each skill contains achieves a multi-step subgoal. The Planner must be more robust, however, because many plans may require a step away from the goal (in a Hamming distance sense) in order to achieve the goal at some later time.

3.4 The Planner

With the basic architecture of the Organizer in place, we can now focus on planning, which is the main function of the Organizer. Planning consists of observing the current state of the environment, receiving a desired goal state from an outside source (such as a user), and formulating an ordered list of sentences which move the state of objects to the goal state when executed. Planning can be likened to a search process which must find an ordered list of sentences which accomplish the goal state and minimize some analytic criteria for the plan.

The following functions are essential to the Planner:

1. A structured search method including subgoal determination.
2. A cost function to be optimized.
3. An evaluation function which predicts the cost of following a particular search direction and other methods for focusing the search.

Acquisition of new information (i.e. exploration of the environment) may also be included when a plan becomes to costly.

3.4.1 A structured search method.

To decide on a structured search method for the Organizer, let us examine a few alternatives.

Exhaustive Search

In the initial formulation for the Organization level of an Intelligent Machine, Valavanis [Val86] forms an optimal plan $\mathbf{P}^*$ to achieve a given goal using the following procedure:

1. For each input command, formulate all possible ordered strings of primative events.
2. Determine which ordered strings are compatible by a table lookup. Compatible ordered strings have pairwise events which do not conflict.
3. Develop augmented ordered activities (strings) by inserting one repetitive primative event at a time between all positions of an ordered string.
4. Reject incompatible augmented ordered activities.
5. Find a complete plan which accomplishes the goal and minimizes the Entropy of the system.

This method is an exhaustive search technique which must develop all possible plans before those which achieve the goal can be filtered out. In a case study performed by Valavanis on plan formulation by a maintanence robot in a Nuclear Power Facility, for a typical user input it was found:

1. The number of unordered strings to be evaluated was 262,143.
2. The number of ordered strings to be evaluated was NOP = 17,403,456,152,414,460.
3. The number of ordered strings to accomplish all user commands was therefore 29*NOP, since there were 29 user commands.

As Valavanis states, these numbers are already huge and not realistic even though they do not include the augmented ordered activities.

Neural Network search

As described in previous sections, a neural network such as a Boltzmann Machine is an efficient environment to perform searches which optimize some analytic measure. The Boltzmann Machines presented search for sentences which minimize an Energy measure manifested in the weights of the network. Each of the sentences contain one actor, one action, one or zero direct objects and one or zero indirect objects. The number of nodes in the network, denoted $\|\mathcal{L}\|$, equals

$$\|\mathcal{L}\| = \mathbf{M} + \|\mathbf{QQ}\| + \|\mathbf{A}\| + \|\mathbf{AV}\| + \|\mathbf{V}\| + \|\mathbf{VD}\| + \|\mathbf{D}\| + \|\mathbf{VI}\| + \|\mathbf{I}\|$$

For a system with 10 objects each in 4 states, 5 actors, 5 actions, 10 direct objects and 10 indirect objects, the number of nodes in the network is:

$$\|\mathcal{L}\| = 40 + 780 + 5 + 25 + 5 + 50 + 10 + 50 + 10 = 975$$

Of these nodes, 70 are *visible* and must be actively searched. The other 905 nodes are *hidden* and are the pairwise connection nodes which are asserted automatically depending on the state of the visible nodes. The number of weights in the network is $\frac{975^2-975}{2} = 474825$. A network of this size is quite reasonable for search and updating.

Expanding a Boltzmann machine to allow for the minimization of complete plans would involve an expansion of the number of nodes and weights. Let us assume that the maximum length of a plan is 10 sentences. Further, let us assume that each sentence in the plan must be represented explicitly for the search to occur. Then a first estimate for the number of nodes for the above system is 9750, with 700 visible nodes. Since the number of states is $2^{numberofvisiblenodes}$, the number of states for this system would be 2^{700} which is tremendous. Also, the number of weights for this system is 47482500, an unwieldy amount considering current memory capacities. A weight matrix of this size also forces an extremely slow learning procedure.

Graph Search

A technique which lends itself well to the Organizer planning problem is Graph Search. Starting at a particular state which corresponds to a place in a graph, the search decides which arc to follow in order to examine other places. The arcs dictate the cost from one place to the next. The objective of the search is to find the minimum cost path from the start place to the goal place. Typical graph representations and graph search algorithms are A^* [HNR68], Means-End Analysis [NE65, NS72] and AND-OR Graphs [Nil71].

The knowledge structure contained in the Organizer decomposes well to a graph. The places in the graph are correspond to object states which are the condition and effect portions of rules. The arcs are formed by the sentence portion of the rule. Searching the graph corresponds to moving from the initial state to the goal state by means of binary sentence execution. The cost of a particular path through the graph is a function of the complexity of a rule and the probability of the rule going to the desired next state. Since all paths do not have to be searched, this technique is much better than exhaustive search. Using a good place evaluation function further reduces the search size so that it performs much better than the neural network search described above.

Graphical decomposition of rules

The following section provides a formal method for the decomposition of rules in the rule store into graph form, and a method for searching the graph based on the A^* algorithm. Proceeding sections will detail sections of this algorithm, as necessary. It should be noted that the "rules" used here can be extended to include skills developed by the goal-directed Boltzmann Machine.

Before presenting the algorithm for graphical decomposition of rules, the following functions must be defined:

1. Let $\mathbf{G_P}(\pi, \mathbf{A})$ represent the directed graph of all possible plans formed by decomposing the rules in $\mathbf{R}$. π denotes the set of all places and $\mathbf{A}$ represents the set of all arcs.

2. Let NAMEP(π_k) be the set of all place names for a subset $\pi_k \in \pi$.

3. Let NAMEA(A_k) be the set of all arc names for a subset $A_k \in \mathbf{A}$.

4. Let PLACEN(NAME) be the place with name NAME.

5. Let ARCN(NAME) be the set of arcs with name NAME.

6. Let PLACEA(A_k) be the set of places directed from arc A_k.

7. Let ARCP(π_k) be the set of arcs directed from place π_k.

8. Let CREATEP(NAME) create a new place and assign it name NAME.

9. Let NEWARC(π_k,SENTNAME,H,P,π_l) create a new arc from place π_k to place π_l, assign it name SENTNAME, complexity H and effect probability P.

10. Let NEWPLACE(π_k, SENTNAME, H, P, PLACENAME) create a new place named PLACENAME and create a new arc from place π_k to the new place, assign it name SENTNAME, complexity H and effect probability P.

The following algorithm constructs a directed graph of places and arcs from decomposing the rules in the rule store:

```
1. Let $R_{gen} = (\forall R_j \epsilon \mathbf{R} : \# \epsilon \Xi_j)$.
2. Let $R_{spec} = \neg R_{gen}$.
3. For all $R_i \ \epsilon \ R_{spec}$:
   (a) If $\Xi_i \epsilon$ NAMEP($\pi$) then:
       i. Let $\pi_k$ = PLACEN($\Xi_k$).
       ii. If $\Sigma_i \ \epsilon$ NAMEA(ARCP($\pi_k$)) then:
           A. Let $A_k$ = ARCP($\pi_k$) $\wedge$ ARCN($\Sigma_i$).
           B. If $\Upsilon_i \ \epsilon$ NAMEP(PLACEA($A_k$)) then go to 3.
           C. If $\Upsilon_i \ \epsilon$ NAMEP($\pi$) then NEWARC($\pi_k$, $\Sigma_i$, $\mathbf{H}_i$, $\mathbf{P}^i_\Upsilon$,PLACEN( $\Upsilon_i$)). Go to 3.
           D. else NEWPLACE($\pi_k$, $\Sigma_i$, $\mathbf{H}_i$, $\mathbf{P}^i_\Upsilon$,$\Upsilon_i$). Go to 3.
       iii. Else NEWPLACE($\pi_k$, $\Sigma_i$, $\mathbf{H}_i$, $\mathbf{P}^i_\Upsilon$,$\Upsilon_i$). Go to 3.
   (b) ELSE:
       i. Let $\pi_k$ = CREATEP($\Xi_i$).
       ii. If $\Upsilon_i \ \epsilon$ NAMEP($\pi$) then NEWARC($\pi_k$, $\Sigma_i$, $\mathbf{H}_i$, $\mathbf{P}^i_\Upsilon$,PLACEN( $\Upsilon_i$)). Go to 3.
       iii. Else NEWPLACE($\pi_k$, $\Sigma_i$, $\mathbf{H}_i$, $\mathbf{P}^i_\Upsilon$,$\Upsilon_i$). Go to 3.
```

For each specific rule in the rule store, the above algorithm assigns a new place for each cause or effect that has not yet been encountered, and builds an arc from the cause to the effect of a rule. The places are given the same name as the cause/effect string, and the arc is given the binary sentence as its name. The arc is also given the complexity of the rule and the effect probability in order to determine the cost of a particular path while searching.

The algorithm above provides an explicit representation for the graph formed by the rules in the rule store. Many search algorithms do not need an explicit representation. Instead, these methods develop a graph implicitly as the search proceeds. An example of such an algorithm is the A^* search technique. The implicit representation is extremely useful when considering the decomposition of generalized rules. In order to create places for generalized rules, one must instantiate all the possible values that can occur when replacing a #, and create a place for each one. An implicit representation would allow the search technique to generate only applicable instantiations for generalized rules.

Given a Rule store $\mathbf{R}$, a Complexity model $\mathcal{B}_{\mathrm{c}}$, a goal directed Boltzmann Machine $\mathcal{B}_{\mathrm{g}}$, a maximum complexity difference θ_E, a maximum allowable path cost MAXCOST, an initial set of object states $\mathbf{Q_0}$ and a user-defined goal state $\mathbf{Q^*}$, let us define the set ACTIVE as the set of places to be expanded and the set CLOSED as the set of already expanded places. Then, the A^* algorithm PLAN proceeds as follows:

PLAN Algorithm

1. Let $\pi_{expand} = \pi_0 =$ CREATEP($\mathbf{Q_0}$).

2. Let $\pi^* =$ CREATEP($\mathbf{Q^*}$).

3. Determine $R_{active} = (\forall R_i \in \mathbf{R} : \Delta(\text{NAMEP}(\pi_{expand}),\Xi_i) = 0)$

4. Generate all places with arcs directed from π_{expand} using the algorithm above if $R_i \in R_{active} \wedge R_i \in R_{spec}$.

5. If $R_i \in R_{active} \wedge R_i \in R_{gen}$ then INSTANTIATE(NAMEP(π_{expand}),R_i). This creates the instance of the general rule R_i which applies to the given state. Then generate arcs and places as above.

6. Add this set of places to ACTIVE.

7. Move π_{expand} to CLOSED.

8. For each $\pi_k \in$ ACTIVE, assign a cost value by:

   (a) Compute $\mathbf{H_c(NAMEA(\pi_k))}$, the complexity of the rule dictated by the Complexity model.

   (b) If $\| \mathbf{H_c(NAMEA(\pi_k))} - \mathbf{H_k} \| \geq \theta_E$ then there is a large diffence between the complexity model computation and the current rule complexity. For computing the cost using evaluation function $f()$, assume the higher complexity value.

   (c) Else, compute cost using f() with $\mathbf{H_k}$.

9. Find $\pi_i \in$ ACTIVE such that $(\forall \pi_k \in \text{ACTIVE} \mid \pi_i \leq \pi_k)$. This is the place with the minimum cost.

10. If $\pi_i = \pi^*$ then DONE. A path has been found. Convert the arcs in this path to binary sentences in Σ, and then to sentences in S. This set of sentences forms the plan $\mathbf{P} \in S^*$. Send $\mathbf{P}$ to Coordination level for execution.

11. If the cost of $\pi_i >$ MAXCOST then enter goal-directed exploration if allowed. Else, return FAILURE.

12. Else, let $\pi_{expand} = \pi_i$. Go to 3.

The A^* algorithm develops a graph implicitly be expanding found places denoted by the ACTIVE set until the goal is reached. It should be noted that decomposing the task into subgoals could significantly reduce the search space for the A^* algorithm. Since subgoal determination is not a trivial task, research must be done on this topic in the context of the Organizer.

Under certain conditions the A^* algorithm can find the minimum cost path to the goal state if one exists. These conditions are dependent upon the place evaluation function. Before we can discuss possible evaluation functions, it is necessary to detail the cost function which must be optimized by the search.

3.4.2 Cost of a plan

The analytic measure which describes the performance of the Intelligent Machine is the Entropy of the system. It is the goal of the Intelligent Machine to minimize this measure, which reflects the execution of a highly certain plan by highly precise means. As the Entropy increases at the Organization level, the certainty of the plan decreases. As it increases at the Execution level, the precision of the control decreases. It is the responsibility of the Organization level to develop a plan which has low Entropy with respect to execution and certainty of success.

The Organizer maintains the complexity of execution $\mathrm{H_c(s)}$ of a sentence s for given object states $\mathbf{Q}$. This value, stored in the rule, is an Entropy measure. Also captured in a rule is the probability that the execution of s under conditions Ξ causes effect Υ. This value is denoted P_Υ. As discussed above, these values can be assigned to arcs in the graphical decomposition of the rule store. We must find a cost function which promotes the search of paths which have high probability effects and have low complexity measures.

Denoting the cost of an arc A_k for rule R_k by $C(A_k)$, one such measure is:

$$C(A_k) = \varphi ln^2(P_\Upsilon^k) + (1-\varphi)\mathrm{H_c}(\mathbf{s_k})$$

The first half of the summation forces search on highly probable rules. The second half forces search based on the Entropy of execution. The constant $0.0 \leq \varphi \leq 1.0$ allows a tradeoff between these two values.

3.4.3 Evaluation Functions

An evaluation function is employed to estimate the past cost and future performance of a search at a given place in a graph. Typical evaluation functions are given by:

$$f(\pi_k) = g(\pi_k) + h(\pi_k)$$

where f(π_k) is the evaluation of place π_k, g() is an estimate of the past cost and h() is an estimate of future performance. Let $h^*(\pi_k)$ be the actual cost from place π_k to the goal state. it has been shown [Nil80] if:

$$h(\pi_k) \leq h^*(\pi_k)$$

for every place $\pi_k \in \pi$, then the solution generated by A^* search will be the minimum cost path. Such a search technique is said to be *admissible*.

Research must be done to find admissible evaluation functions for the Organizer. Two simple ones are:

1. Best First: Set h to 0 for all places. Let $g(\pi_k)$ equal the total arc cost of the path from π_0 to π_k.

2. Locality of Effect: Under the locality of effect proposition, the execution of a sentence effects the states of the direct and indirect objects. At each place π_k, enumerate the number of objects which must change state. This determines the minimum number of rules which must be executed. Multiply this value by a constant to scale to an arc cost, and set $h(\pi_k)$ equal to this value.

Other methods may also be incorporated to focus the search along particular paths. As described previously, the goal-directed Boltzmann machine can be used to generate subtask sentences which have a high certainty of achieving desired object state changes. These sentences can reduce the search space by focusing the examination on arcs which possess the subtask sentence name. The effect of this focusing procedure on the search space must be examined in depth in the context of the A^* algorithm.

3.4.4 Acquisition of New Information

When developing a plan, it may become apparent that the least cost path found exceeds a maximum cost threshold, even though the goal place has not yet been found. If this is the case, the Organizer may select to explore the environment in a similar method as PLAY. Instead of randomly selecting sentences to execute in the environment, the Organizer can generate sentences through the goal-directed Boltzmann Machine. By providing desired state changes as input to $\mathcal{B}_g$, the Planner can receive a set of sentences to execute which have a high certainty of moving the system toward the goal. After executing these sentences, the Planner can replan from the current state to the goal.

3.5 Summary

This section described the architecture and operation of the Organizer, which is the level that accomplishes abstract planning and decision making in the Intelligent Machine. The Organizer contains the following features:

- Symbolic rules which form a syntactic environmental effect model.
- Connectionist networks which assign semantic interpretation to the symbolic rules.
- An emergent framework which develops and modifies its information through experience.
- Rule effects modeled probablistically.
- Generalization of both symbolic and semantic knowledge.
- Goal-directed behavior, search and planning.

The blueprint presented details the Rule store, the Complexity model, the Generalizer, the goal-directed Boltzmann Machine and the Planner. The first two mechanisms maintain the basic knowledge structures which are required to model the actions of the Intelligent Machine on its environment. The advanced mechanisms abstract information from the basic mechanism to facilitate the planning procedure. The Planner uses a graph search technique to find the set of ordered sentences to execute to achieve a given goal. Each of these mechanisms is developed within the context of Saridis' Intelligent Machine, which provides an analytic framework for the development of the architecture.

4 Further Research

Research Goals

The following list presents a set of research topics spawned by the architecture and methodology proposed in this paper:

1. The complexity model must be modified to bound the connection weights between 0.0 and 1.0. This will form a uniform model with the goal-directed Boltzmann Machine.

2. The training technique must be verified and full experimentation must be done on the complexity model. Testing must be done to characterize the generalization capabilities of the network.

3. Other methods of generalization must be developed and experiments must be performed with the Generalizer. Methods for creating a more robust default hierarchy must be examined.

4. Symbolic learning systems developed by other research should be tested in the context of the Organizer.

5. A training technique must be developed for the goal-directed Boltzmann Machine.

6. Search techniques must be further developed which extract minimum Energy binary sentences from the goal-directed Boltzmann Machine, corresponding to sentences with maximum $K(n)$.

7. A method for developing subgoals in planning must be explicitly designed and experimented with.

8. Several place evaluation functions must be found and tested.

9. The determination of a suitable analytic search criteria combining task complexity and likelihood of success must be addressed.

10. The use of goal-directed behavior as an exploration tool after plan failure must be looked into.

11. Computer simulations must be run to test each of the units separately and as they function together.

12. A case study must be performed in a problem domain which examines the performance of the Organizer.

Proposed Research

This report proposes accomplishing the following research, in addition to the work already completed in this paper:

Given a rule store and a complexity model intact:

1. A training technique will be developed for the goal-directed Boltzmann Machine. The Boltzmann Machine will train on each rule in the store to develop a semantic representation between the binary sentences and the object state changes.
2. Search techniques will be developed for extracting sentences which maximize the Knowledge about the state of the machine. These search techniques must take into account the sparseness of the Machine states in order to be efficient.
3. Methods for forming skills based on subtask determination using the goal-directed Boltzmann machine will be designed and tested.
4. Place evaluation functions for search during planning will developed to reduce the search space.
5. Development of an analytic cost criteria during search must be experimented with. An example of one, which provides a simple tradeoff between complexity and probability of effect, is presented in this paper.
6. A case study using the above five research goals will be performed to test their functionality.

Future work

Eventually, the following areas must also be researched and developed:

1. Allowing the architecture to dynamically expand to accommodate the introduction of new objects or states in the environment.
2. Testing the Organizer in a real-world environment.
3. Abstracting the binary states into predicate calculus relations to increase the generalization capacity of the system.

References

[Ack87] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers, 1987.

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[Alb75] J. S. Albus. A new approach to manipulation control: The cerebellar model articulation controller. *Transactions of the ASME Journal of Dynamic Systems, Measurement and Control*, 97:220–227, 1975.

[And72] J. A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14:197–220, 1972.

[And83] J. R. Anderson. *The Architecture of Cognition.* Harvard University Press, Cambridge, MA, 1983.

[APW88] P. J. Antsaklis, K. M. Passino, and S. J. Wang. Autonomous control systems: Architecture and fundamental issues. In *Proceedings of the American Control Conference*, volume 1, pages 602–607, 1988.

[AR88] J. A. Anderson and E. Rosenfeld. *Neurocomputing: Foundations of research.* The MIT Press, 1988.

[BFKM86] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5.* Addison-Wesley, 1986.

[Blo62] H. D. Block. The Perceptron: a model for brain functioning. I. *Reviews of Modern Physics*, 34:123–135, 1962.

[Boo89] L. B. Booker. Triggered rule discovery in classifier systems. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 265–274, 1989.

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[Car86] J. G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquistion. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[CS67] B. Chandresekaran and D. W. C. Shen. On expediency and convergence in variable-structure automata. *IEEE Transactions on Systems, Science and Cybernetics*, 4:52–60, 1967.

[Day87] D. S. Day. Towards integrating automatic and controlled problem solving. In *IEEE First International Conference on Neural Networks*, volume 2, pages 661–669, San Diego, CA, 1987.

[DC87] L. Davis and S. Coomb. Genetic algorithms and communication link speed design: Theoretical considerations. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 252–256, Cambridge, MA, 1987.

[DD87] C. P. Dolan and M. G. Dyer. Symbolic schemata, role binding and the evolution of structure in connectionist memories. In *IEEE First International Conference on Neural Networks*, volume 2, pages 287–298, San Diego, CA, 1987.

[DeJ86] G. DeJong. An approach to learning from observation. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Volume II*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986.

[DJ75] K. A. De Jong. *An Analysis of the Behavior of a class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[DP86] M. Derthick and D. C. Plaut. Is distributed connectionism compatible with the physical symbol system hypothesis? In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986.

[FB82] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.

[FHN72] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 1972.

[FL69a] K. S. Fu and T. J. Li. Formulation of learning automata and automata games. *Information Science*, 1(3):237–256, 1969.

[FL69b] K. S. Fu and T. J. Li. On stochastic automata and languages. *Information Science*, 1(4):403–419, 1969.

[FN71] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3,4), 1971.

[Fu67] K. S. Fu. Stochastic automata as models of learning systems. In J. T. Tou, editor, *Computer and Information Sciences II.* Academic Press, New York, 1967.

[Fu71] K. S. Fu. Stochastic automata, stochastic languages and pattern recognition. *Journal of Cybernetics*, 1(3):31–49, 1971.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, November 1984.

[GGRG85] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms*, pages 160–168, 1985.

[Gol83] D. E. Goldberg. *Computer-aided pipeline operation using genetic algorithms.* PhD thesis, The University of Michigan, Ann Arbor, MI, 1983.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[GP88] K. Goldberg and B. Pearlmutter. Using a neural network to learn the dynamics of the CMU Direct-Drive Arm II. Technical Report CMU-CS-88-160, Carnegie Mellon University, Pittsburgh, PA, 1988.

[Gro88] S. Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1(1):17–62, 1988.

[Har89] S. Harnad. The symbol grounding problem. In *CNLS Conference on Emergent Computation*, Los Alamos, N.M., May 1989.

[HdM89] L. S. Homem de Mello. *Task Sequence Planning for Robotic Assembly.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1989.

[HdMS88] L. S. Homem de Mello and A. C. Sanderson. Planning repair sequences using the and/or graph representation of assembly plans. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1861–1862, 1988.

[HHNT86] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction.* The MIT Press, Cambridge, MA, 1986.

[Hin86] G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, 1986.

[Hir87] M. W. Hirsch. Convergence in neural nets. In *IEEE First International Conference on Neural Networks*, volume 2, pages 115–124, San Diego, CA, 1987.

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, 4:100–107, 1968.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[Hop84] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81:3088–3092, 1984.

[HS86] G. E. Hinton and T. J. Sejnowski. Learning and relearning in Boltzmann machines. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing Volume I.* The MIT Press. Cambridge, MA. 1986.

[HS87] W. R. Hutchison and K. R. Stephens. Integration of distributed and symbolic knowledge representations. In *IEEE First International Conference on Neural Networks*, volume 2, pages 395–398. San Diego. CA, 1987.

[HT85] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimiziation problems. *Biological Cybernetics*, 52:141–152, 1985.

[Jay57] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4), 1957.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 4598, 1983.

[KM89] R. B. Kelley and M. C. Moed. Knowledge-based robotic assembly system. In G. N. Saridis, editor, *Advances in Automation and Robotics, Vol. 2.* JAI Press, 1989.

[Koh72] T. Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, C-21:353–359, 1972.

[KUIS88] M. Kawato, Y. Uno, M. Isobe, and R. Suzuki. Hierarchical neural network model for voluntary movement with application to robotics. *IEEE Control Systems Magazine*, pages 8–15. April 1988.

[Lip87] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[Lue84] D. L. Luenberger. *Linear and Nonlinear Programming, Second Edition.* Addison-Wesley, 1984.

[Mau84] M. L. Maudlin. Maintaining diversity in genetic search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 247–250, 1984.

[MDM88] M. Marra, T. Dunlay, and D. Mathis. Terrain classification using texture for the ALV. In *Proceedings of the 1988 SPIE Symposium on Advances in Intelligent Robotic Systems*, 1988.

[ME70] J. M. Mendel and K. S. Fu Eds. *Adaptive, Learning and Pattern Recognition Systems.* Academic Press, New York, 1970.

[Min61] M. Minsky. Steps toward Artificial Intelligence. *Proceedings of the Institute of Radio Engineers*, 49, 1961.

[MKSS88] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki. Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, 1:251–265, 1988.

[MP69] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[MPRV87] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, IT-33(4), July 1987.

[MS89] M. C. Moed and G. N. Saridis. A Boltzmann machine for the organization of intelligent machines. In *Proceedings of the NASA Conference on Telerobotics*, Pasadena, CA, January 1989.

[NE65] A. Newell and G. Ernst. The search for generality. In W. A. Kalenich, editor, *Information Processing 65: Proceedings of IFIP Congress 1965*, pages 17–24. Spartan Books, Washington, D. C., 1965.

[New80] A. Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[Nil71] N. J. Nilsson. *Problem-solving Methods in Artificial Intelligence.* McGraw-Hill, 1971.

[Nil80] N. J. Nilsson. *Principles of Aritifical Intelligence.* Tioga Publishing Company, Palo Alto, California, 1980.

[NS72] A. Newell and H. A. Simon. *Human Problem Solving.* Prentice-Hall, 1972.

[NS76] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19:113–126, 1976.

[NT74] K. S. Narendra and M. A. L. Thathachar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4(4):323–334, 1974.

[PM47] W. Pitts and W. S. McCulloch. How we know universals: the perception of auditory and visual form. *Bulletin of Mathematical Biophysics*, 9:127–147, 1947.

[RHW86a] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing Volume I*, pages 318–362. The MIT Press, Cambridge, MA, 1986.

[RHW86b] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Rio87] R. L. Rioli. Bucket brigade performance: I. Long sequences of classifiers. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 184–195, Cambridge, MA, 1987.

[Rio89] R. L. Rioli. The emergence of coupled sequences of classifiers. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 256–264, 1989.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing, Vol. I.* The MIT Press, Cambridge, MA, 1986.

[Ros58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[Ruc87] D. W. Ruck. Multisensor target detection and classification. Master's thesis, AFIT/GE/ENG, Wright-Patterson AFB, Ohio, 1987.

[SA89] L. Shastri and V. Ajjanagadde. A connectionist system for rule based reasoning with multi-place predicates and variables. Technical Report MS-CIS-89-06, University of Pennsylvania, Philadelphia, PA, 1989.

[Sar73] E. D. Sarcerdoti. Planning in a hierarchy of abstraction spaces. In *Third International Joint Conference on Artificial Intelligence*, Stanford, Ca. 1973.

[Sar75] E. D. Sarcerdoti. The nonlinear nature of plans. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, 1975.

[Sar77] G. N. Saridis. Expanding subinterval random search for system identification and control. *IEEE Transactions on Automatic Control*, pages 405–412, 1977.

[Sar79] G. N. Saridis. Toward the realization of intelligent controls. *IEEE Proceedings*, 67(8), 1979.

[Sar88] G. N. Saridis. Entropy formulation for optimal and adaptive control. *IEEE Transactions on Automatic Control*, 33(8):713–721, 1988.

[Sar89] G. N. Saridis. On the revised theory of intelligent machines. In *Proceedings of an International Workshop on Intelligent Robots and Systems*, Tsukuba, Japan, September 1989.

[SG84] G. N. Saridis and J. H. Graham. Linguistic decision schemata for intelligent robots. *Automatica*, 20(1), 1984.

[SG87] J. Y. Suh and D. Van Gucht. Incorporating heuristic information into genetic search. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, Cambridge, MA, 1987.

[SM88] G. N. Saridis and M. C. Moed. Analytic formulation of intelligent machines as neural nets. In *Proceedings of the IEEE Conference on Intelligent Control*, Washington, DC, August 1988.

[Ste81a] M. Stefik. Planning with constraints (MOLGEN: part 1). *Artificial Intelligence*, 16, 1981.

[Ste81b] M. Stefik. Planning with constraints (MOLGEN: part 2). *Artificial Intelligence*, 16, 1981.

[Ste87] Luc Steels. Self-organization through selection. In *IEEE First International Conference on Neural Networks*, volume 2, pages 55–62, San Diego, Ca, 1987.

[Sus88] H. J. Sussmann. Learning algorithms for Boltzmann machines. In *Proceedings of the 27th Conference on Decision and Control*, pages 786–791, Austin, TX, 1988.

[Sut88] R S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[SV88] G. N. Saridis and K. P. Valavanis. Analytical design of intelligent machines. *Automatica*, 1988.

[TH85] D. S. Touretzky and G. E. Hinton. Symbols among the neurons: Details of a connectionist inference architecture. In *Proceedings of the International Joint Conference on Aritificial Intelligence*, Los Angeles, CA, 1985.

[Tou87] D. S. Touretzky. Representing conceptual structures in a neural network. In *IEEE First International Conference on Neural Networks*, volume 2, pages 279–286, San Diego, CA, 1987.

[Val86] K. P. Valavanis. *A Mathematical Formulation for the Analytical Design of Intelligent Machines.* PhD thesis, Rensselaer Polytechnic Institute. Troy, NY, 1986.

[Vam87] T. Vamos. Metalanguages - conceptual model: Bridge between machine and human intelligence. In *Proceedings of the 1st International Symposium on AI and Expert Systems*, pages 237–287, 1987.

[WG89] S. W. Wilson and D. E. Goldberg. A critical review of classifier systems. In *Proceedings of an International Conference on Genetic Algorithms*, pages 244–255, 1989.

[WH60] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON Convention Record*, pages 96–104, New York, 1960.

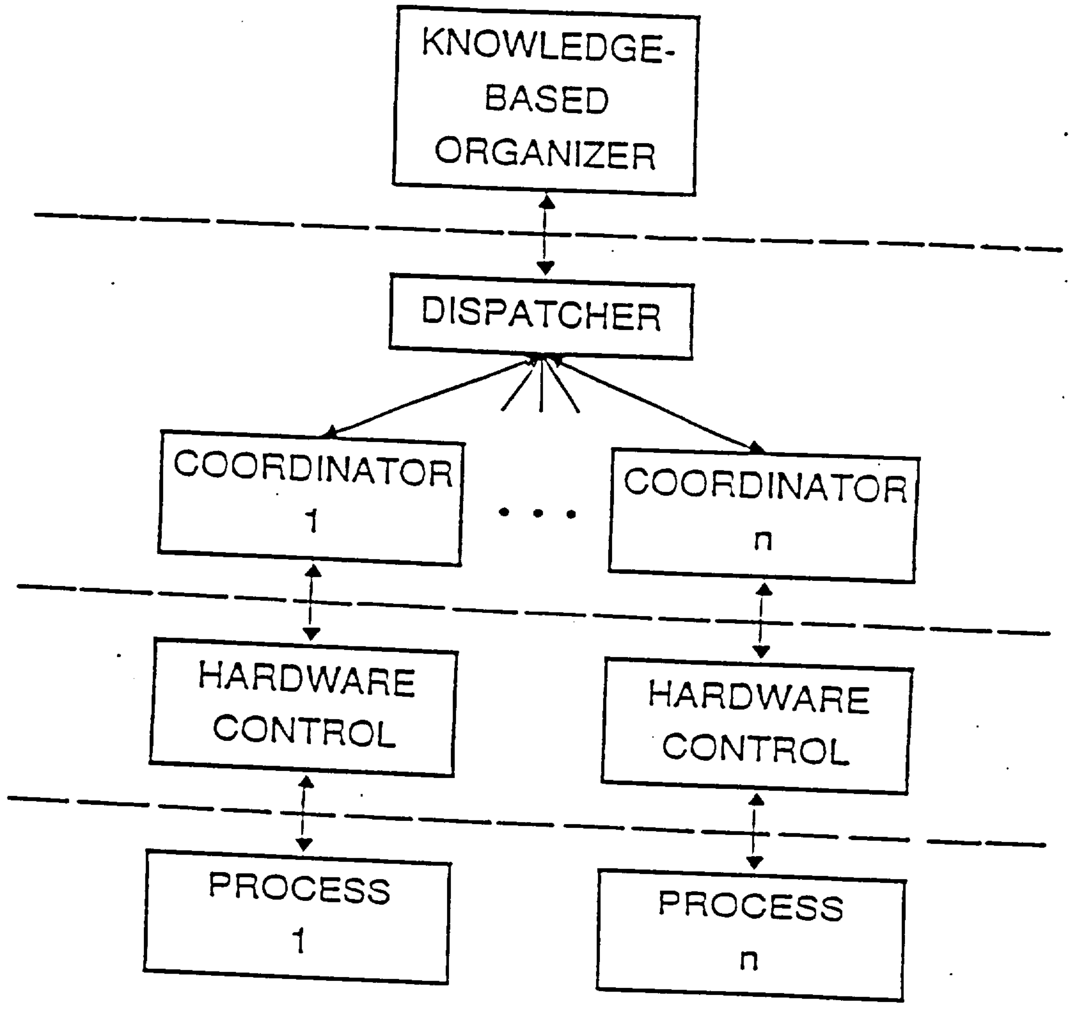
[Wil85] S. W. Wilson. Knowledge growth in an artificial animal. In *Proceedings of an International Conference on Genetic Algorithms*, pages 16–23, 1985.

[Wil87] S. W. Wilson. Hierarchical credit allocation in a classifier system. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 217–220, 1987.

[Wil88] R. J. Williams. Towards a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[WS88] F. Wang and G. N. Saridis. A model for coordination of intelligent machines using Petri nets. In *IEEE Symposium on Intelligent Control*, Washington, D.C., August 1988.

[Zho87] H. H. Zhou. *CSM: A genetic classifier system for learning by analogy.* PhD thesis, The University of Michigan, Ann Arbor, MI, 1987.



Hierarchical Intelligent Control System

Figure 2.1

ORGANIZER ARCHITECTURE

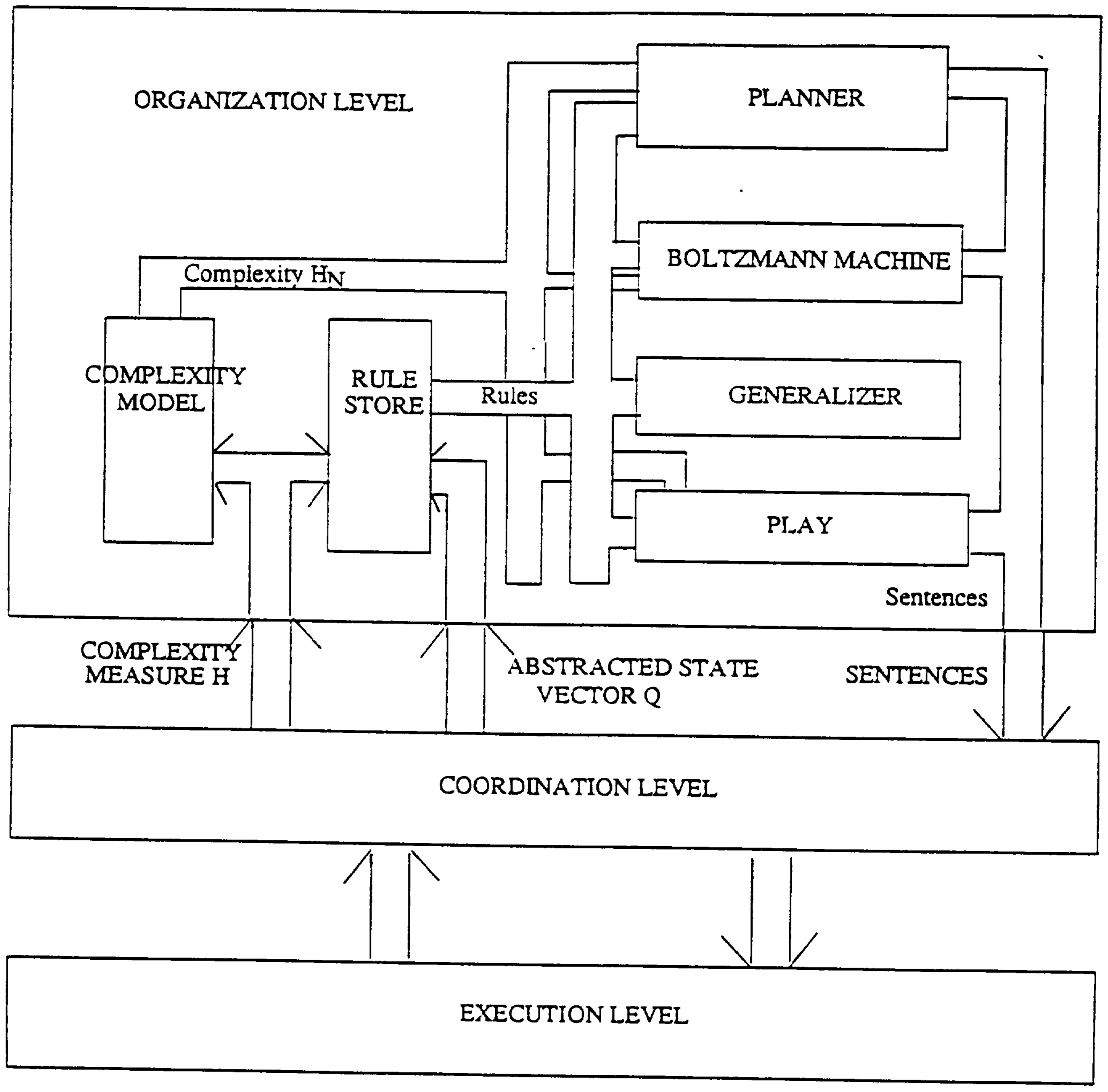


Figure 3.1

NETWORK DIAGRAM AND PARTIAL WEIGHT DESCRIPTION

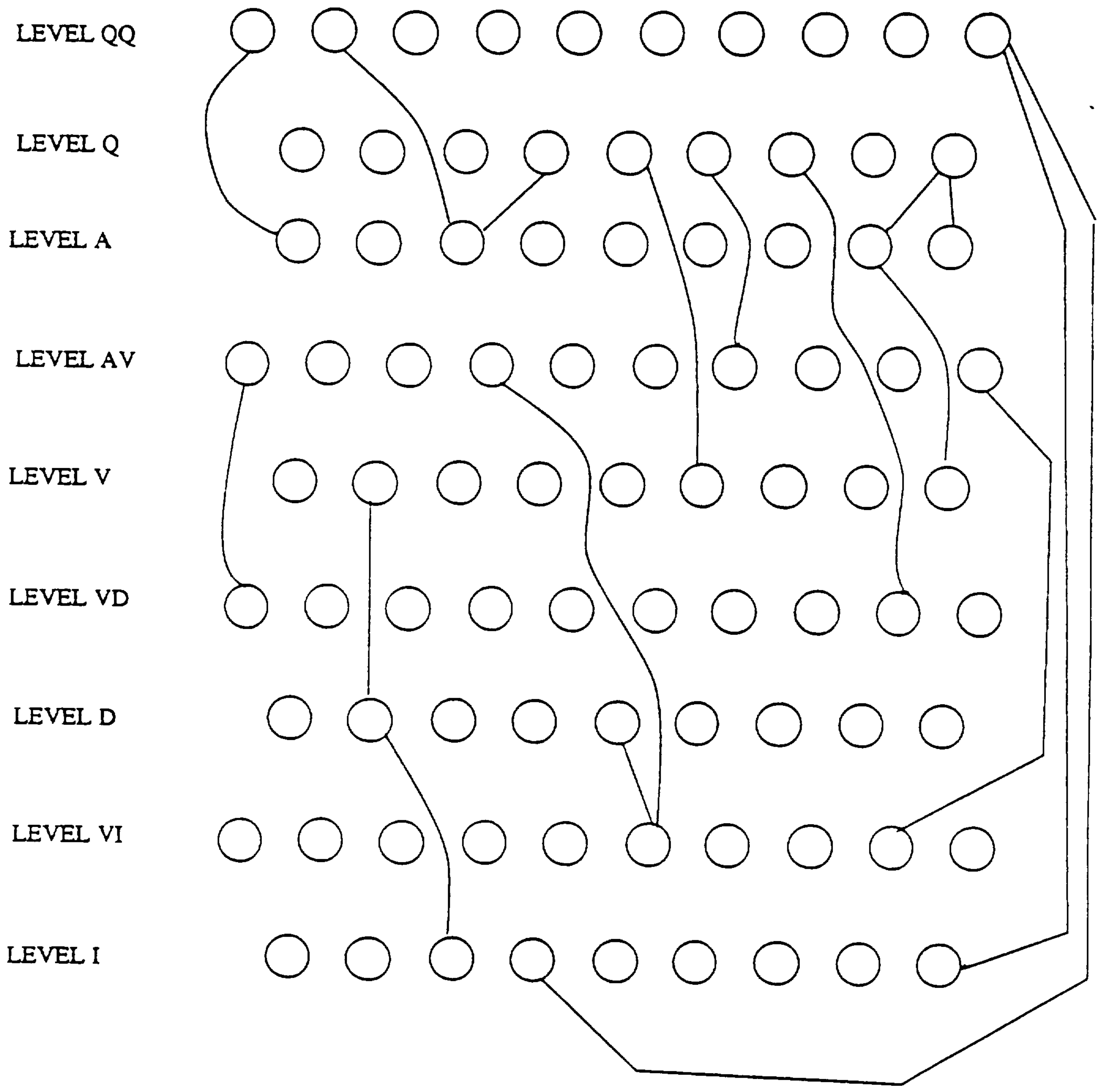


Figure 3.2

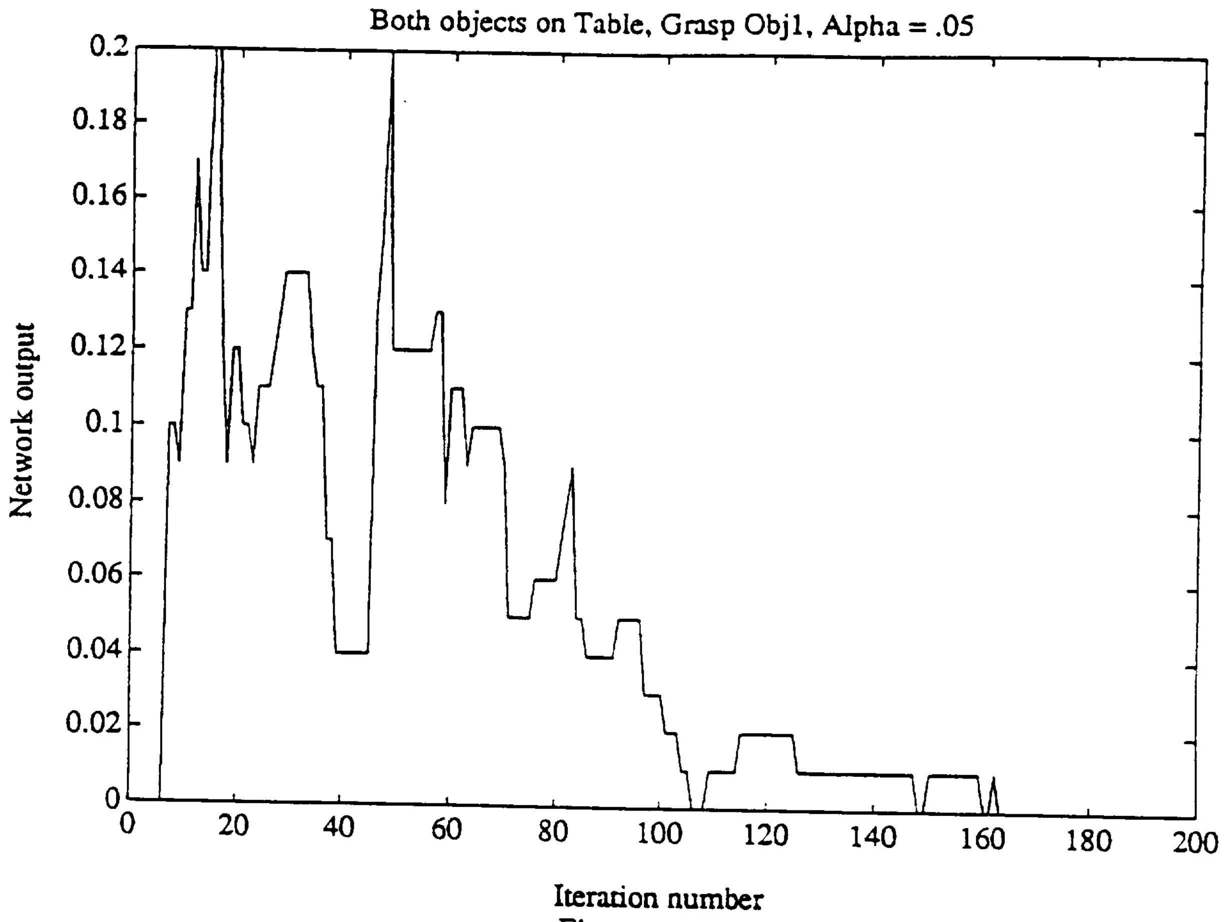


Figure 3.3a

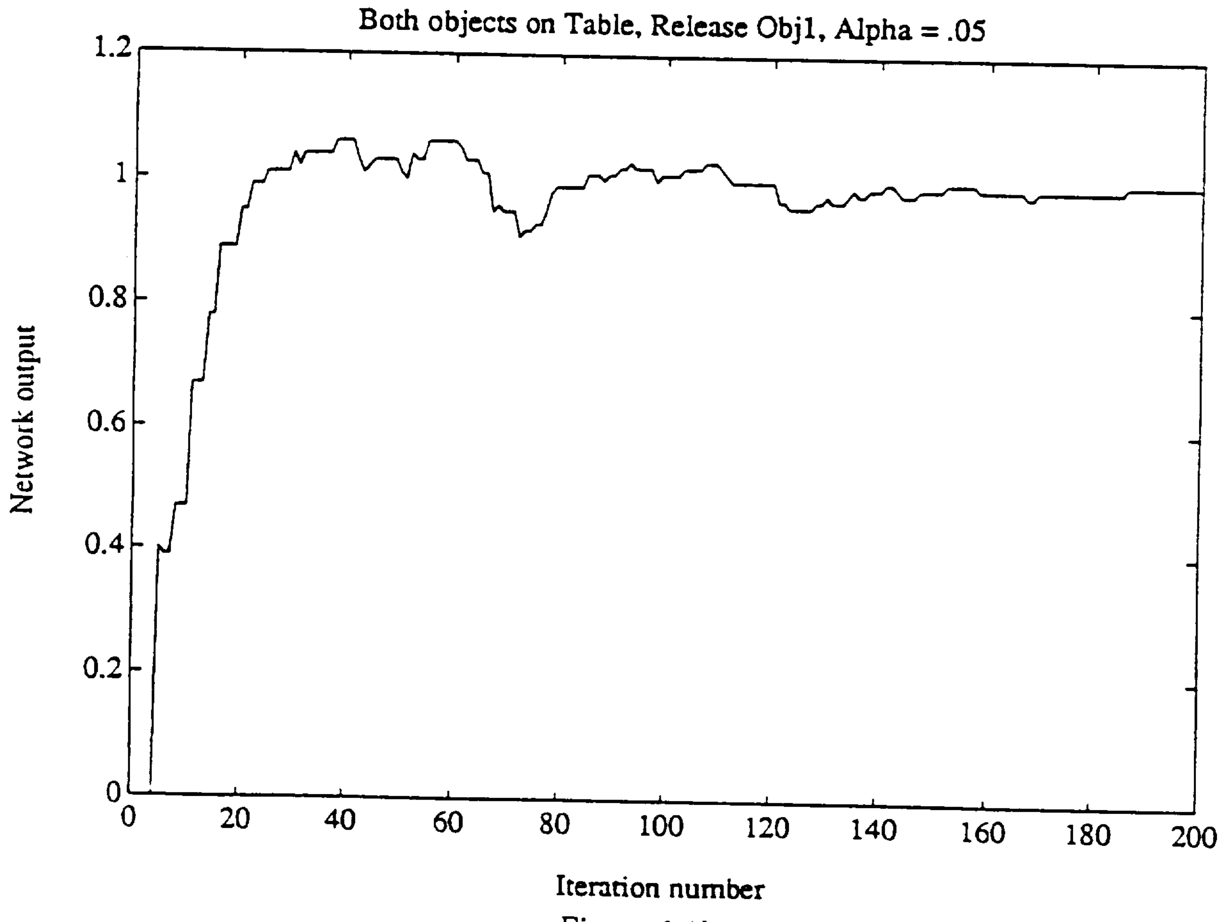


Figure 3.3b

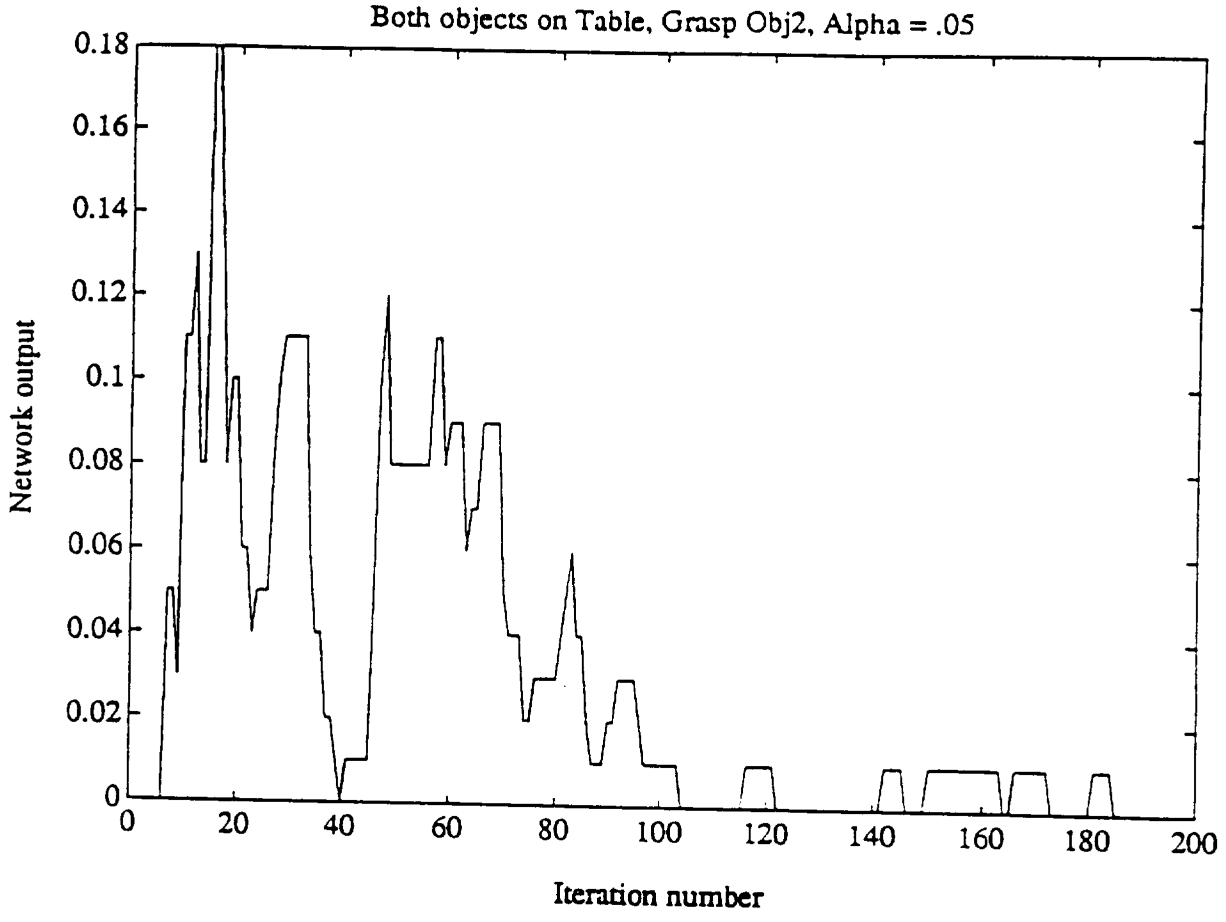


Figure 3.3c

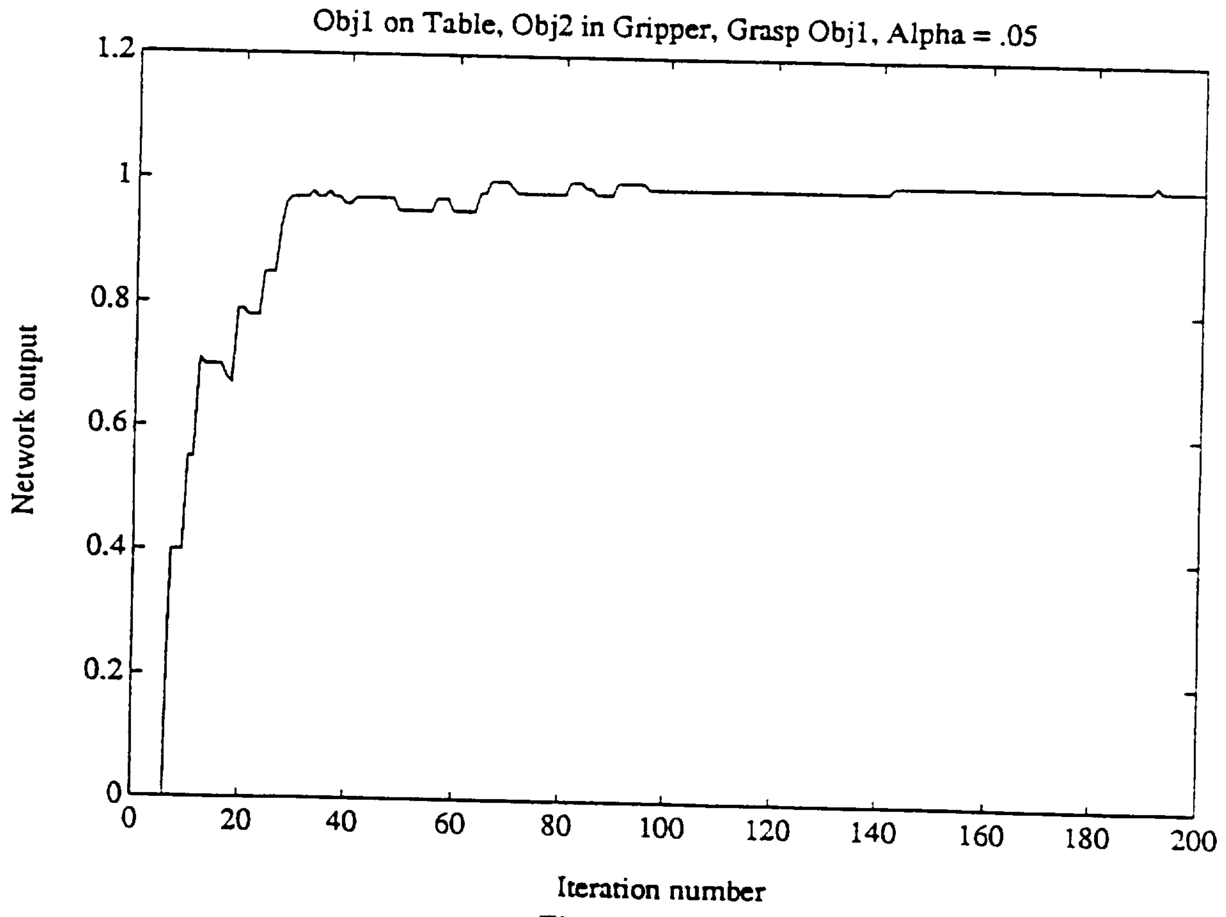


Figure 3.3d

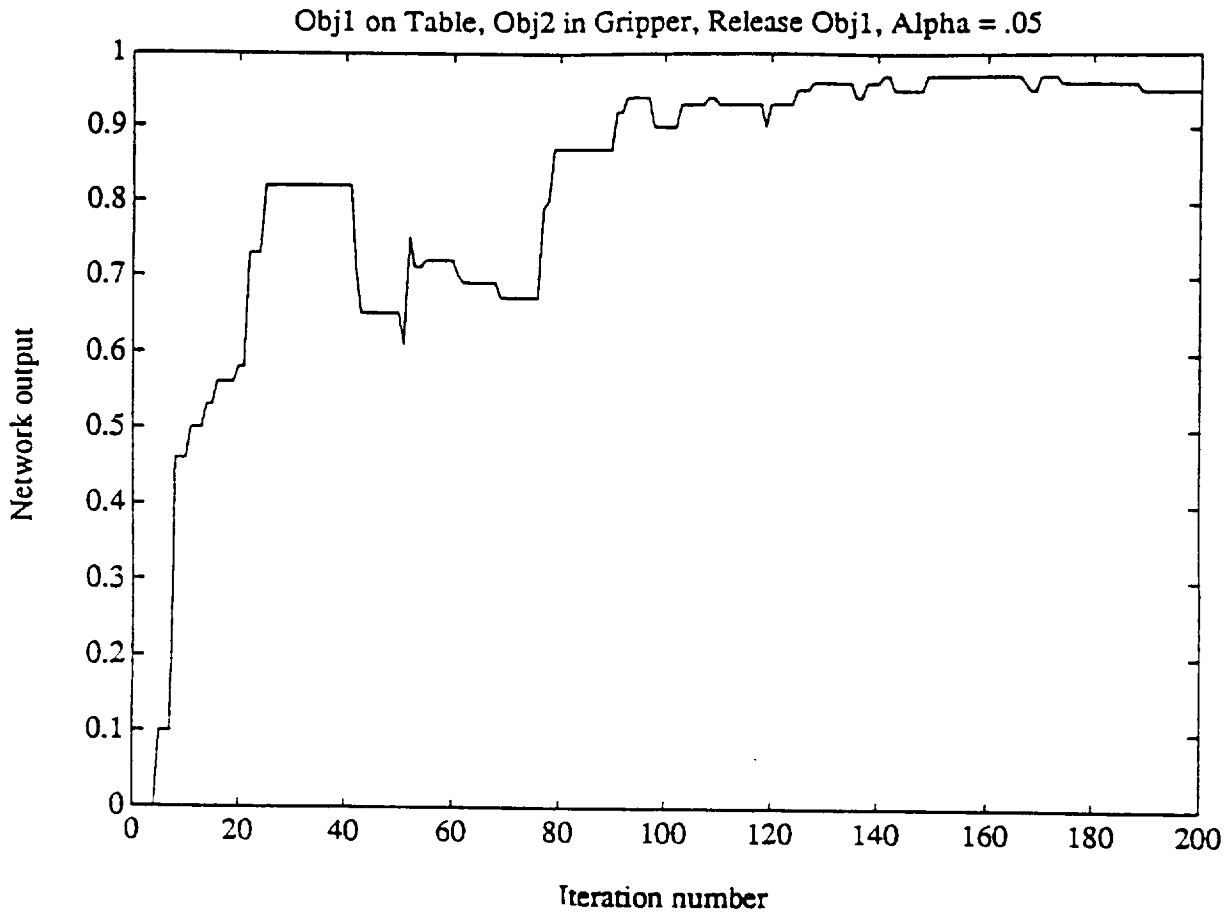


Figure 3.3e

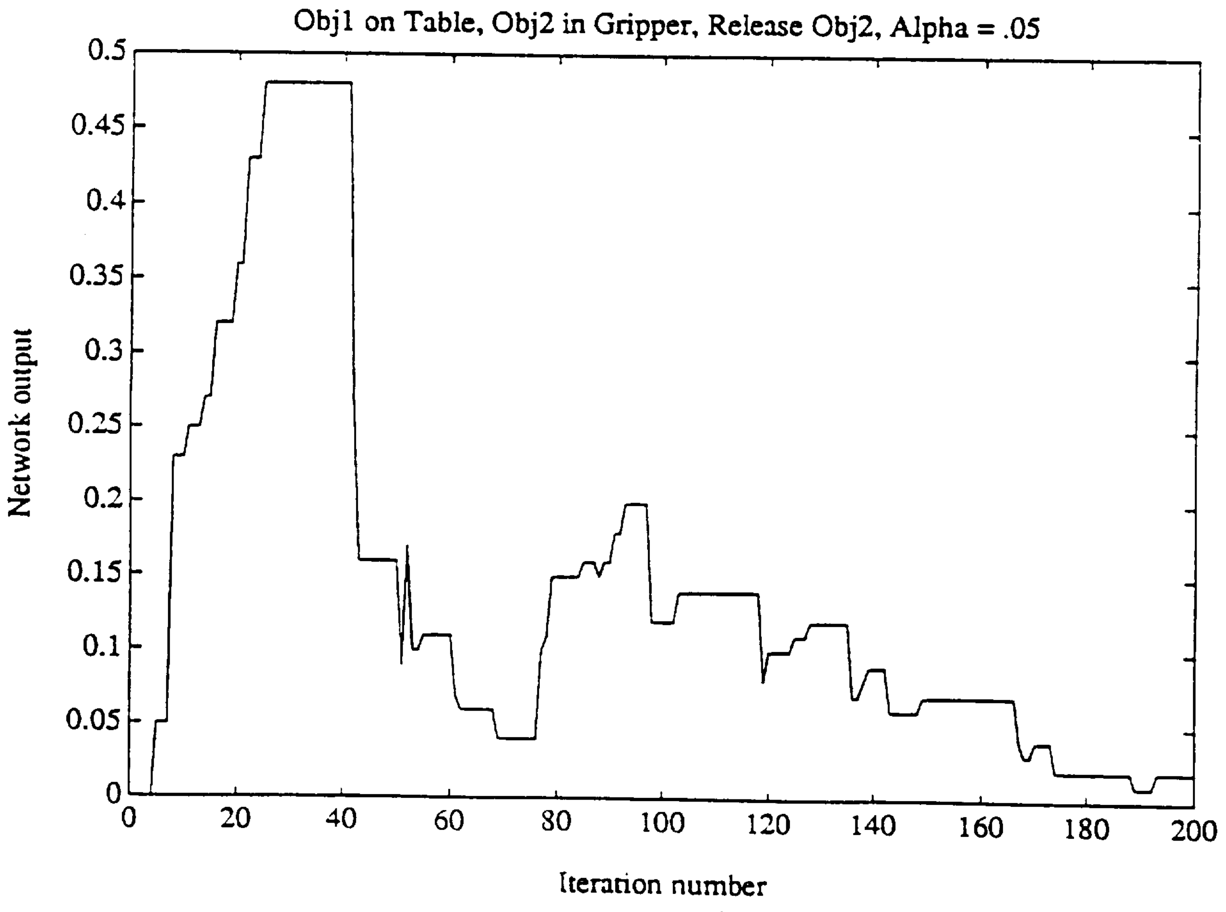


Figure 3.3f

ORIGINAL PAGE IS
OF POOR QUALITY

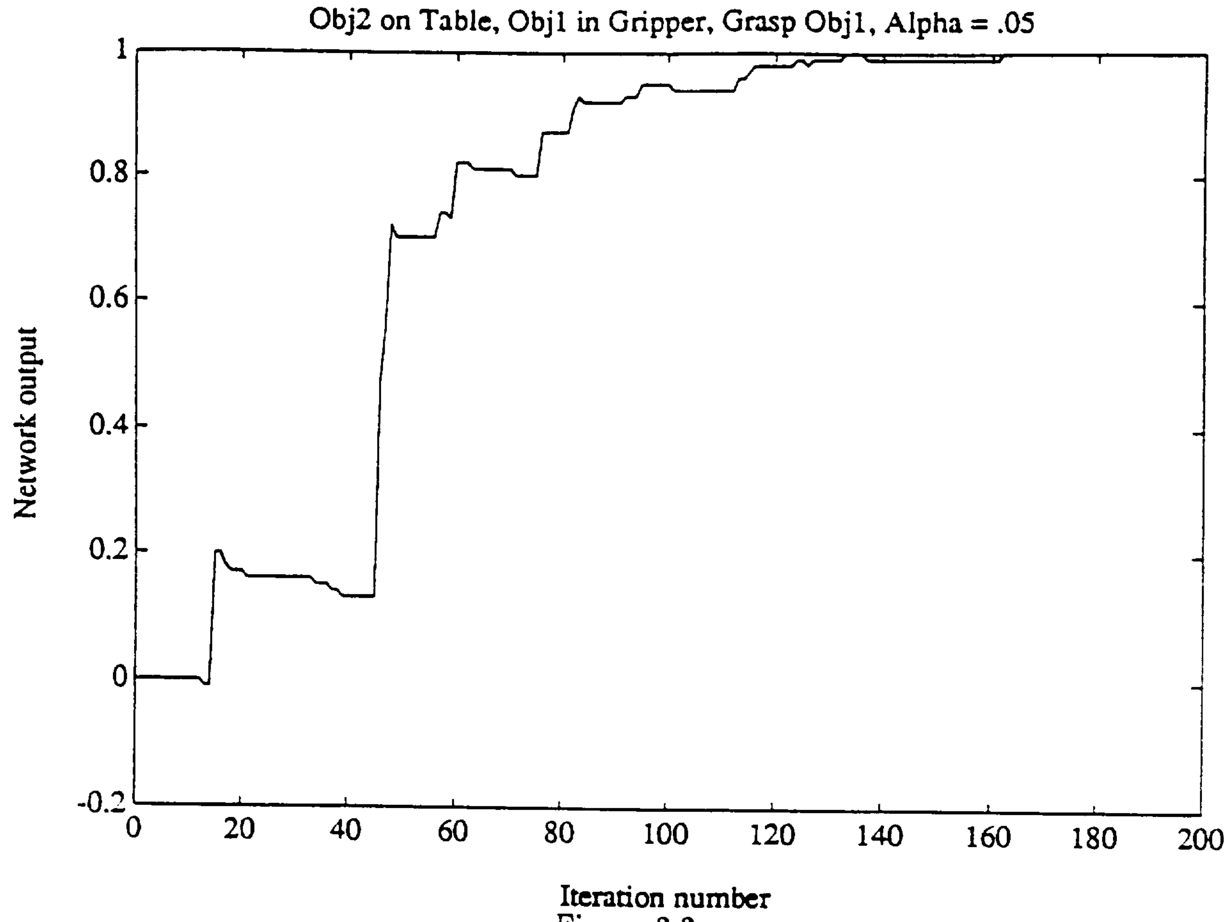


Figure 3.3g

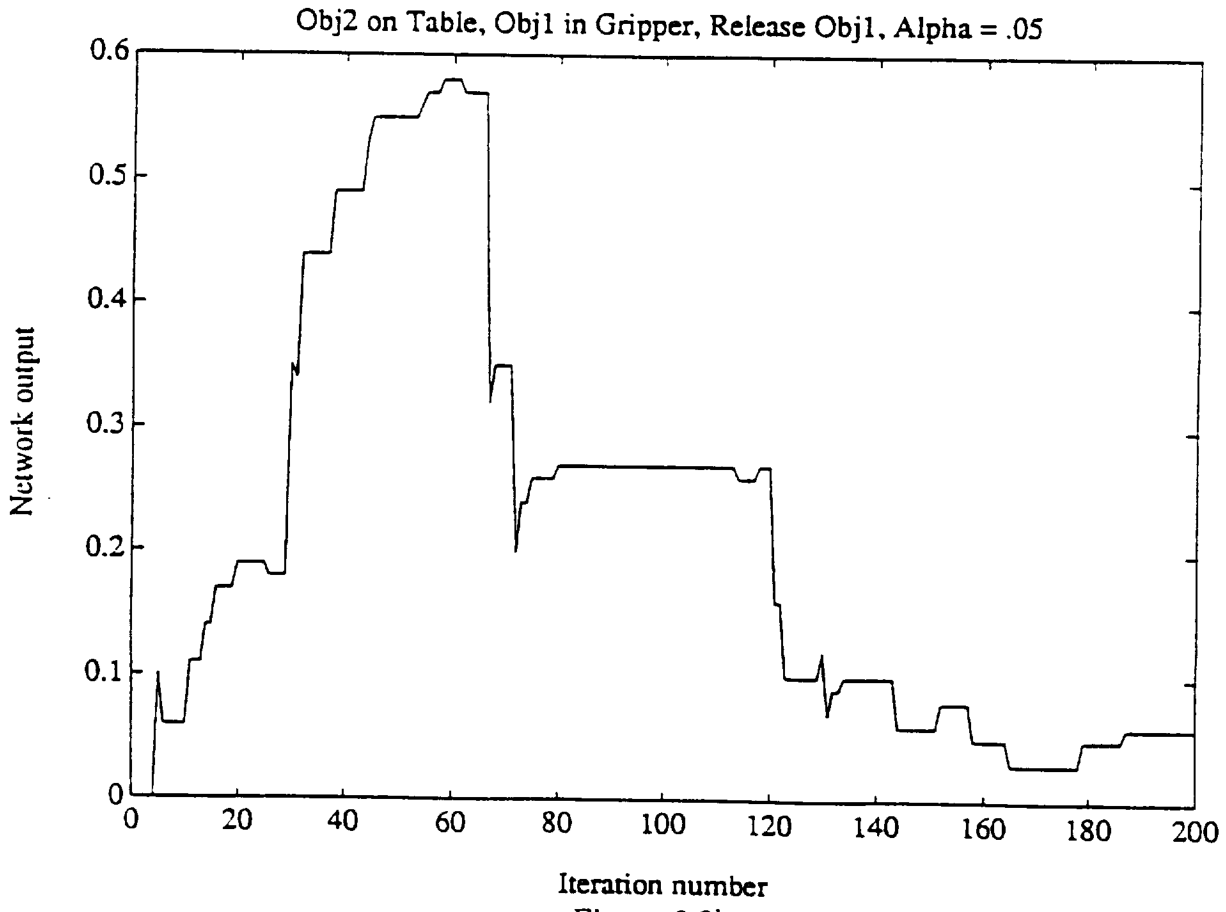


Figure 3.3h

C-2

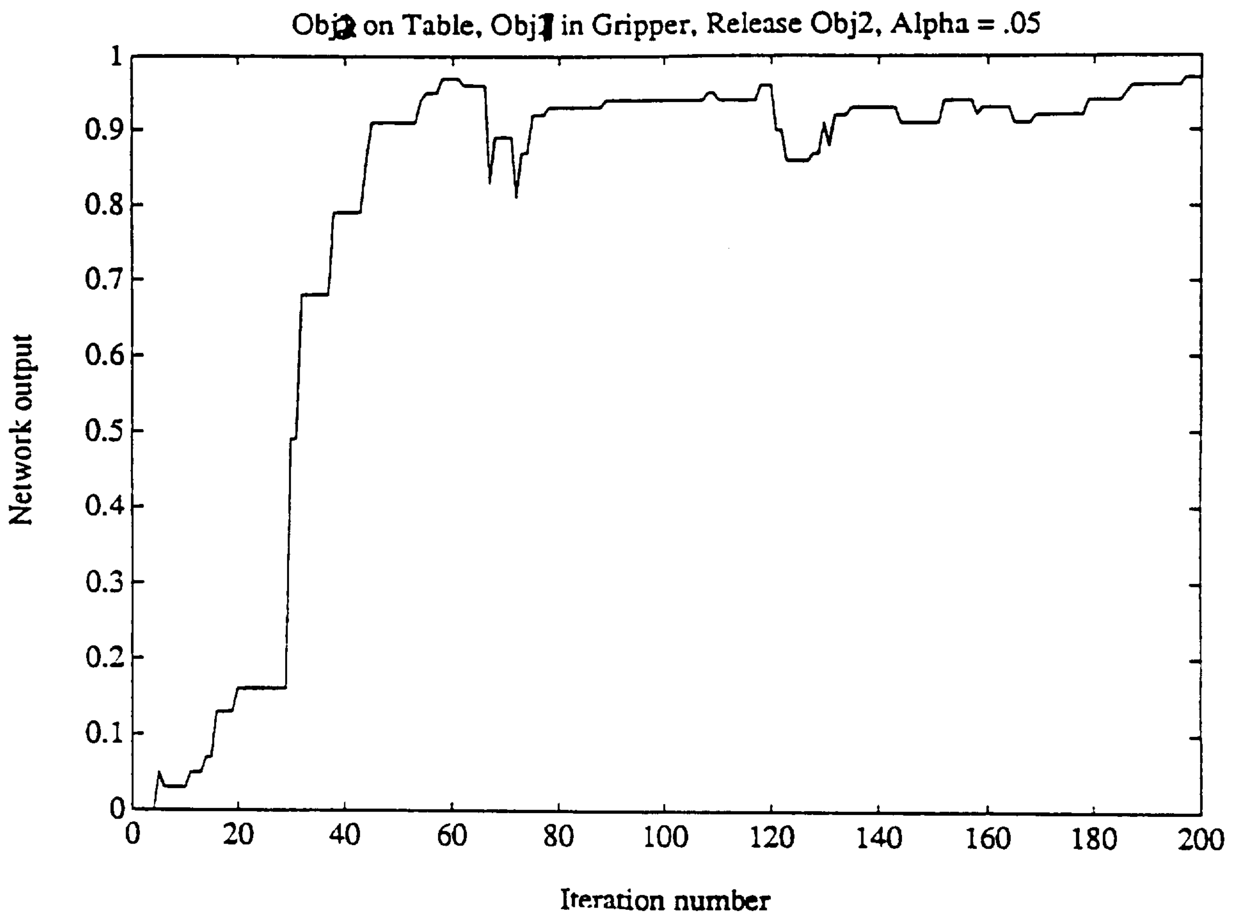


Figure 3.3i

ORIGINAL PAGE IS
OF POOR QUALITY

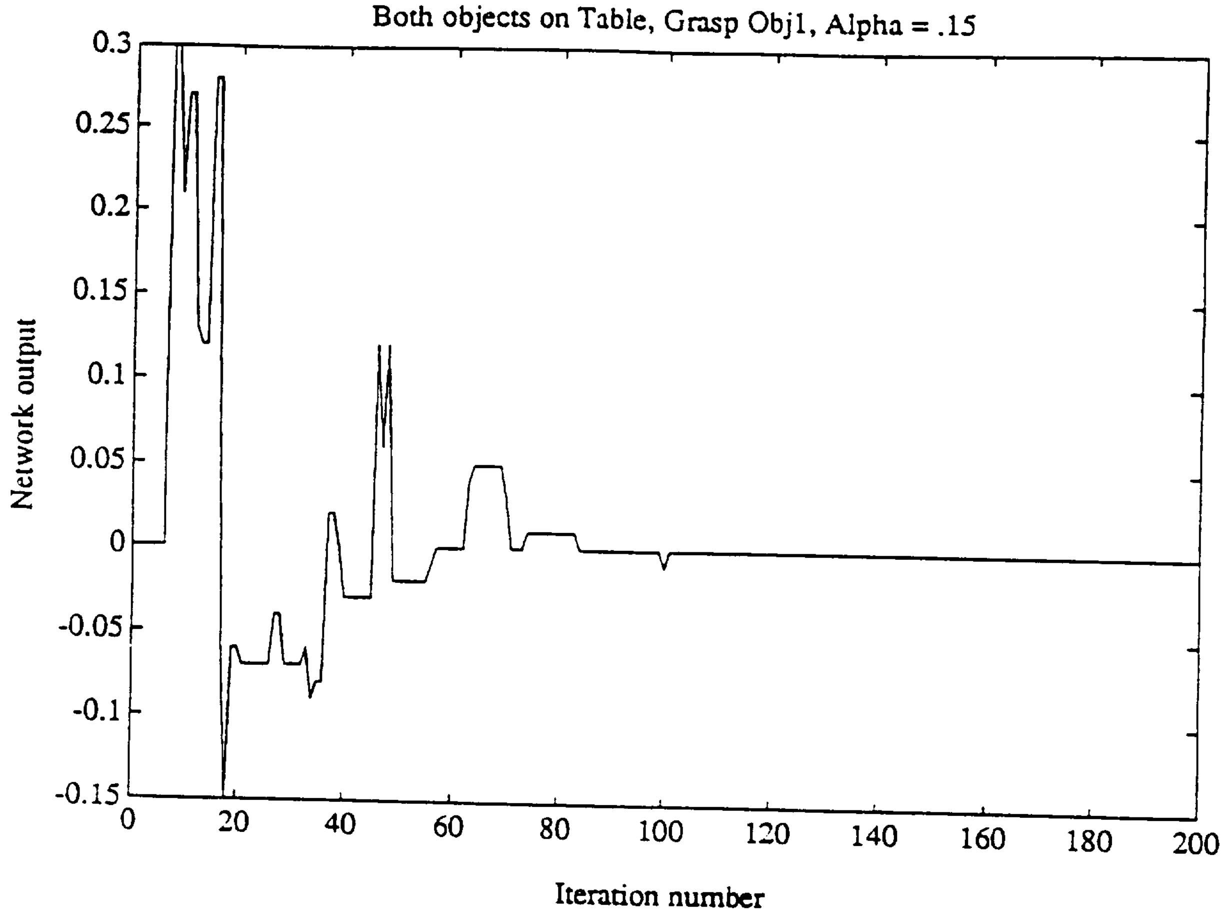


Figure 3.4a

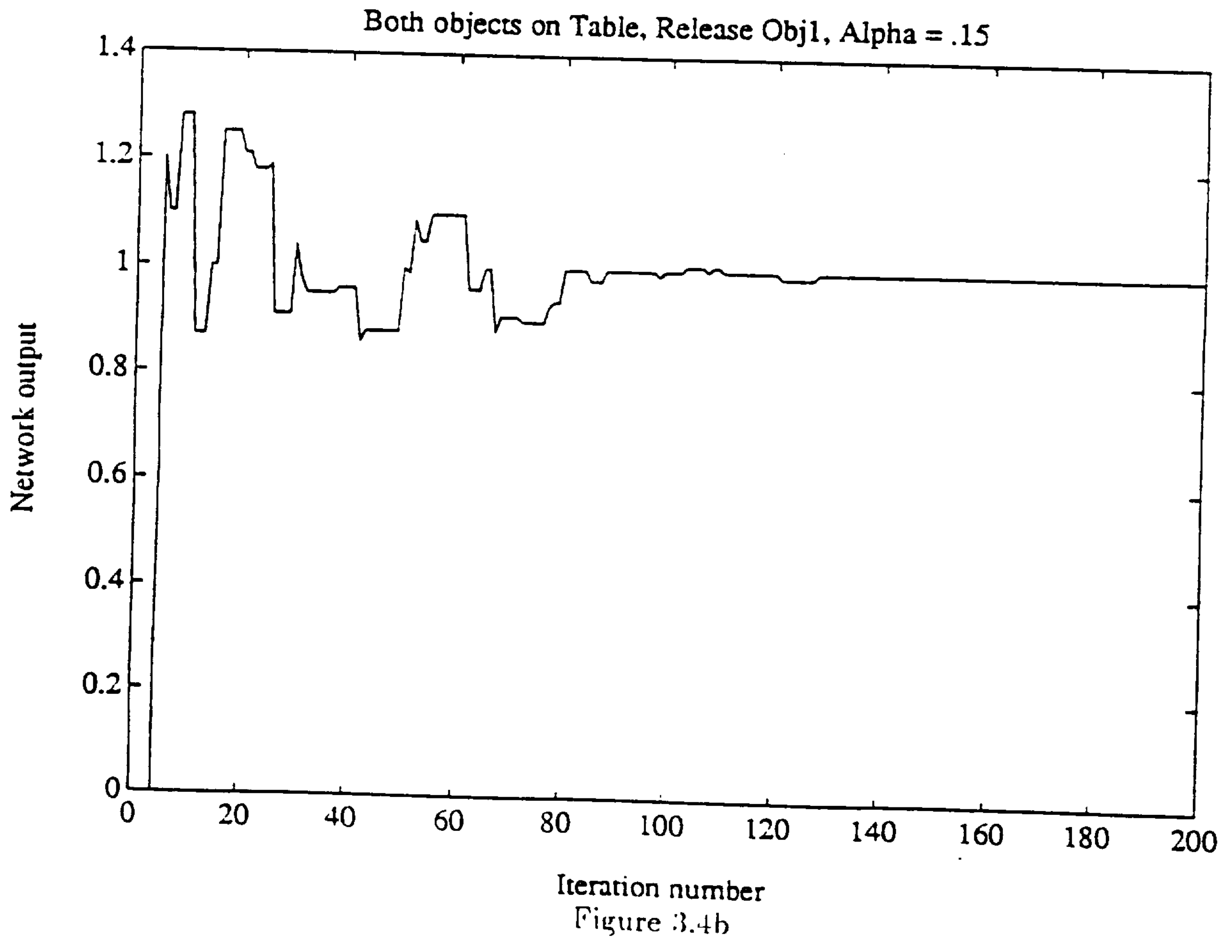


Figure 3.4b

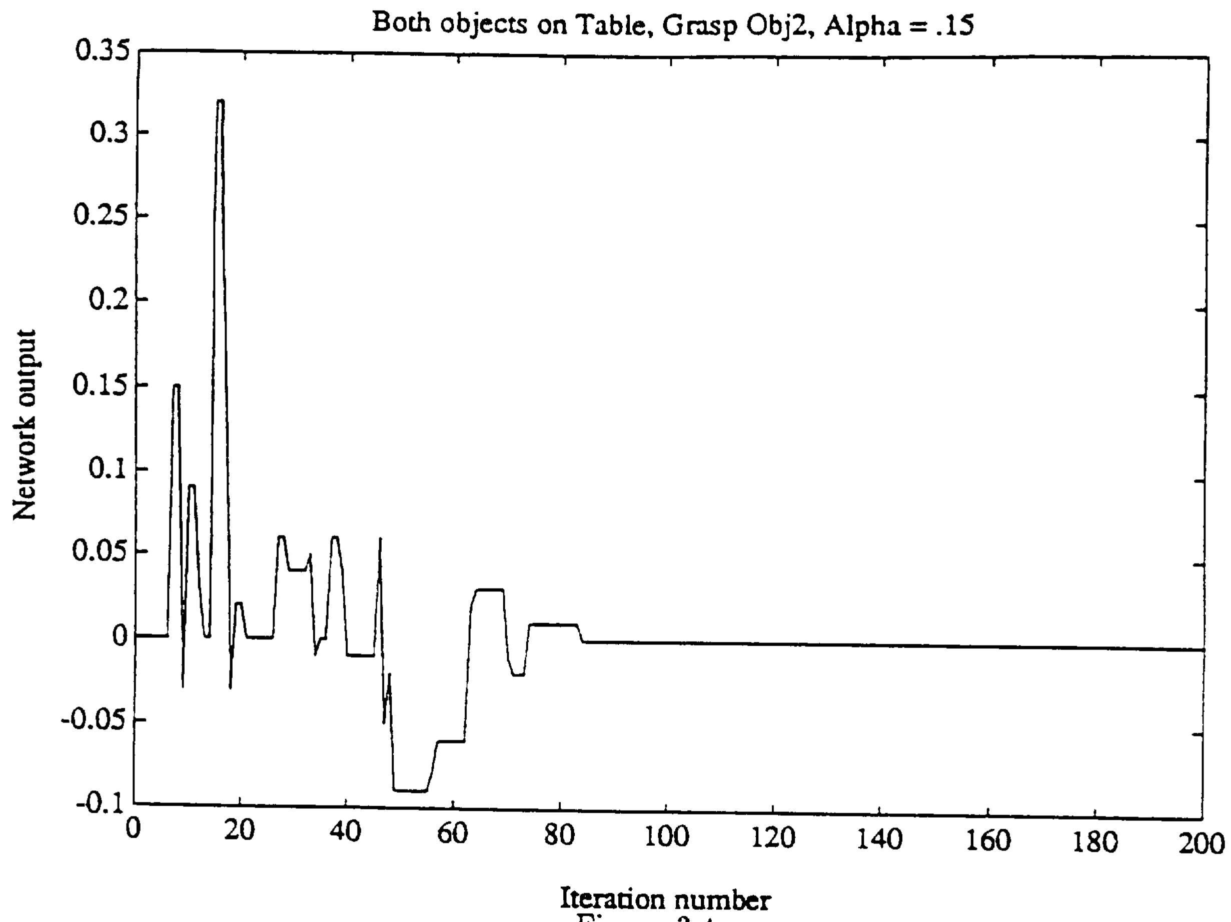


Figure 3.4c

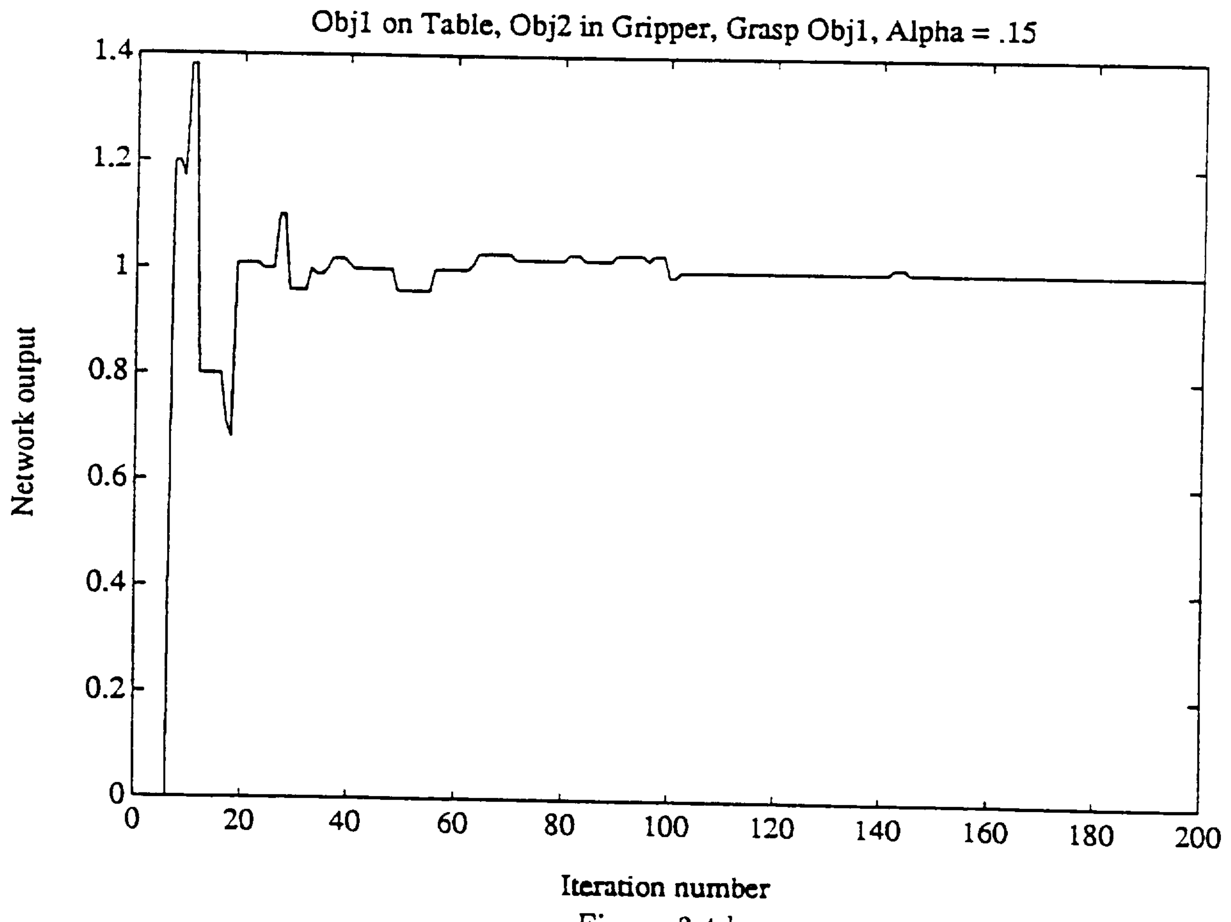


Figure 3.4d

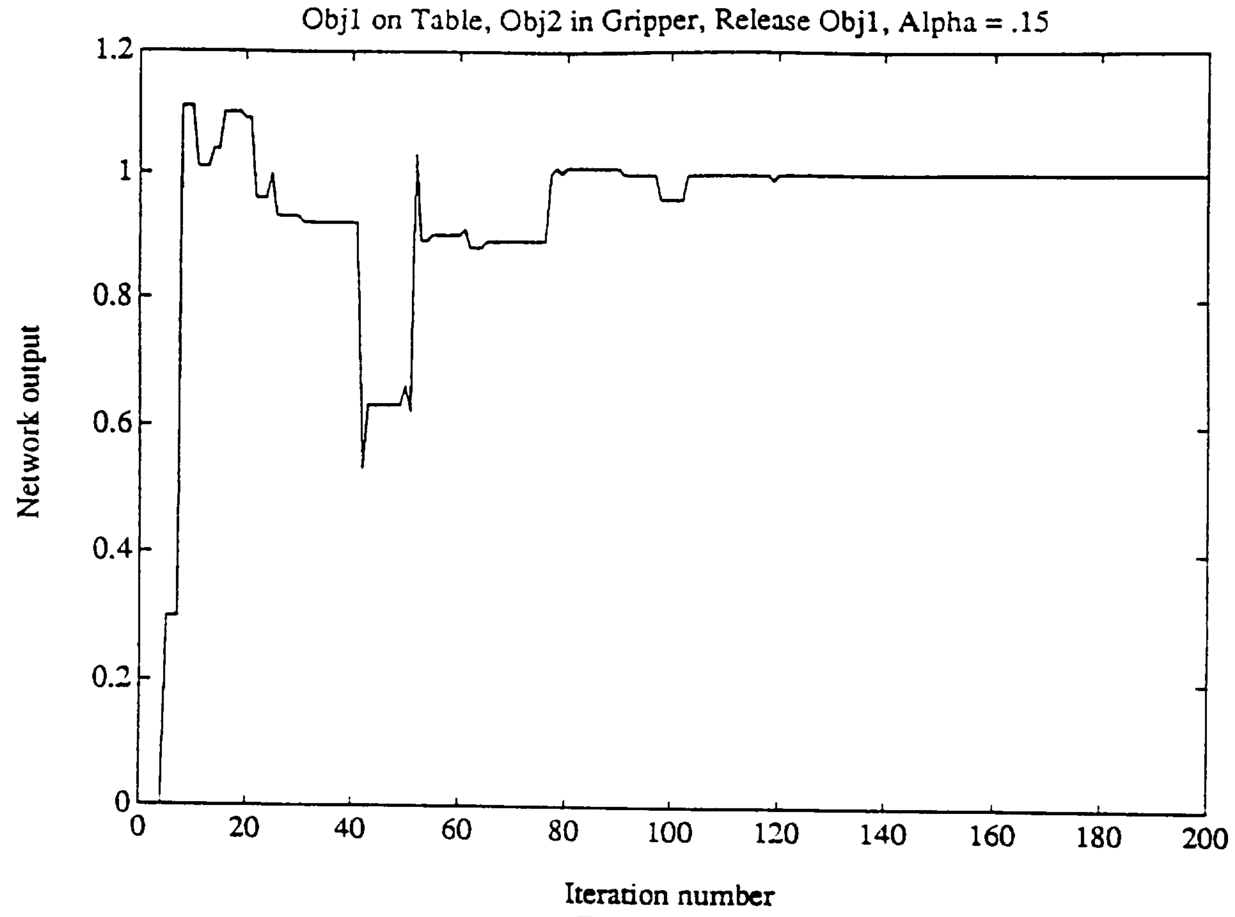


Figure 3.4e

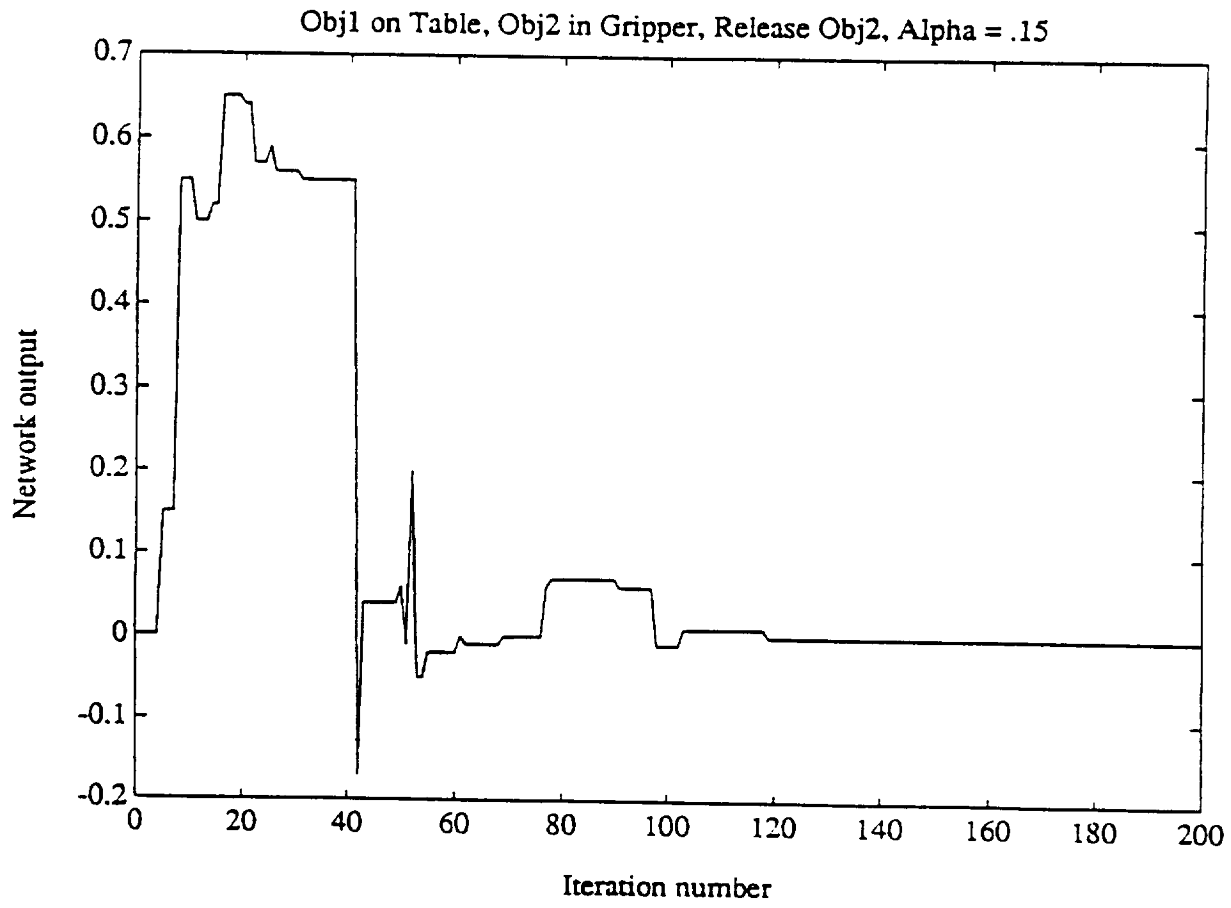


Figure 3.4f

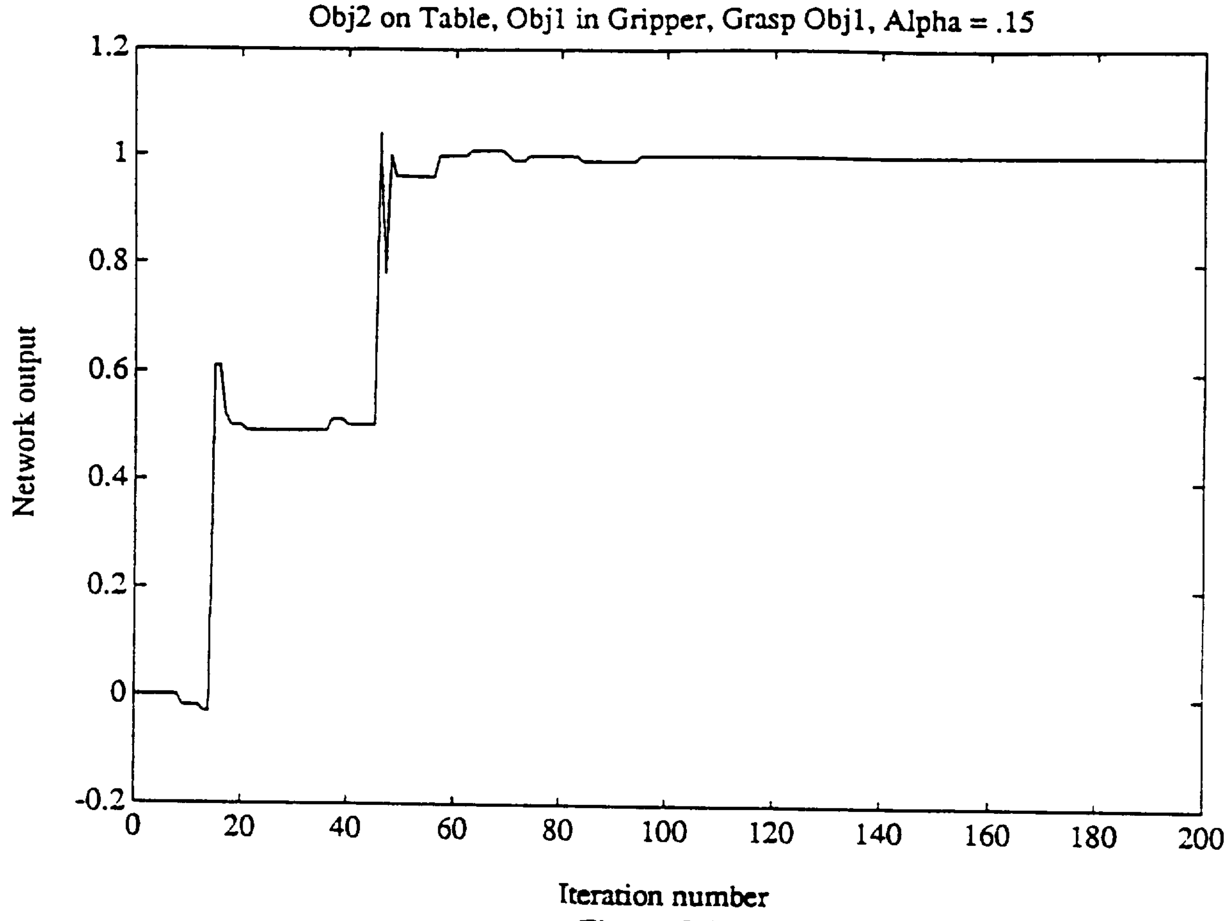


Figure 3.4g

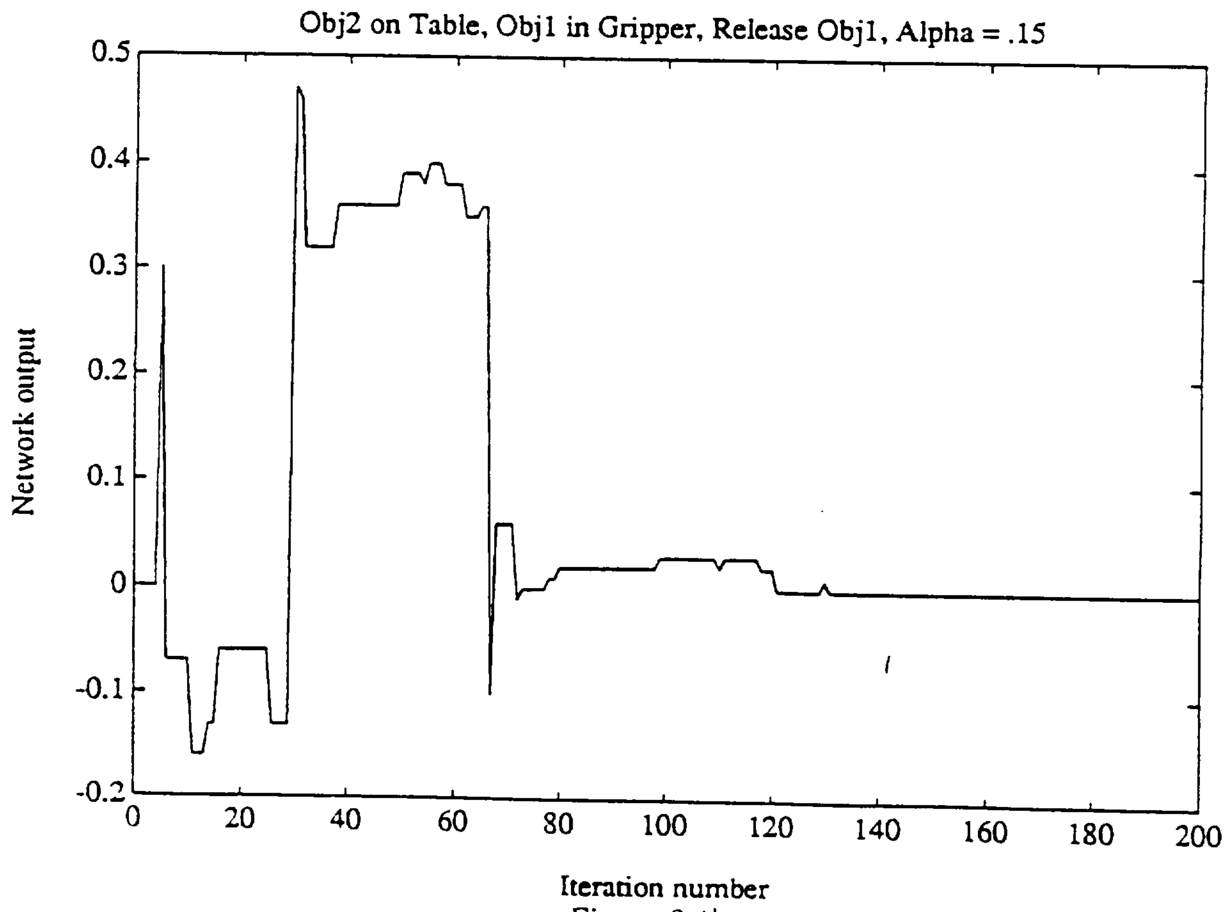


Figure 3.4h

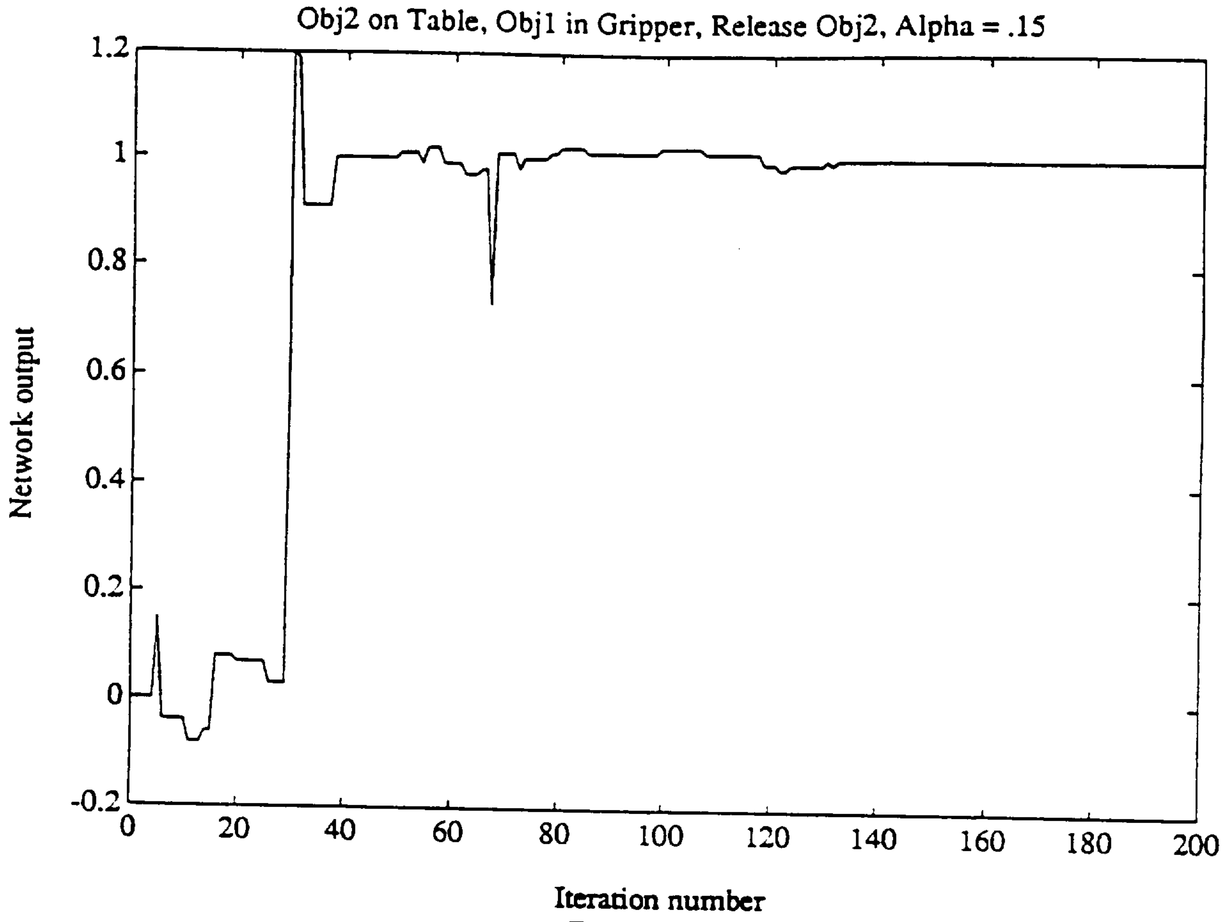

Obj2 on Table, Obj1 in Gripper, Release Obj2, Alpha = .15
Network output
Iteration number
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0
20
40
60
80
100
120
140
160
180
200

Figure 3.4i

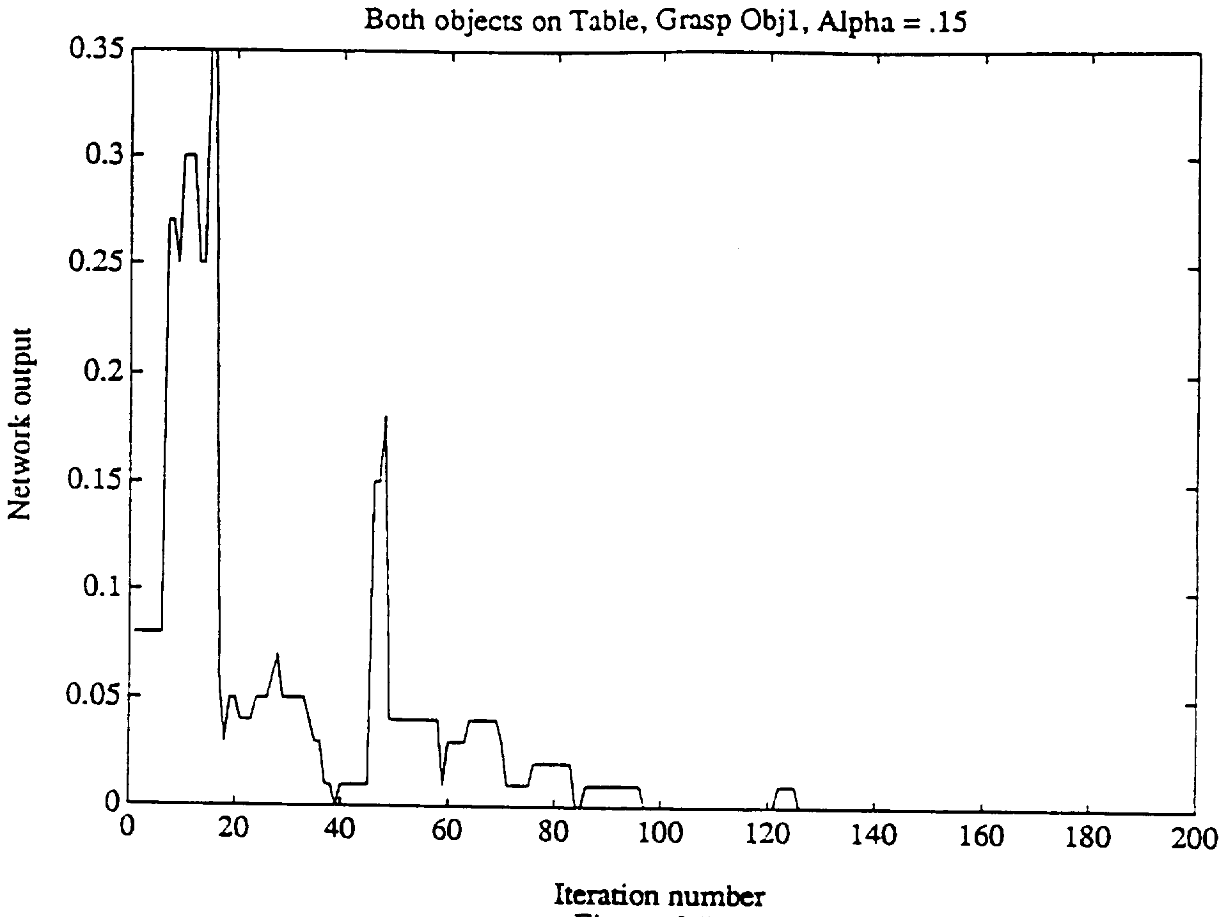


Figure 3.5a

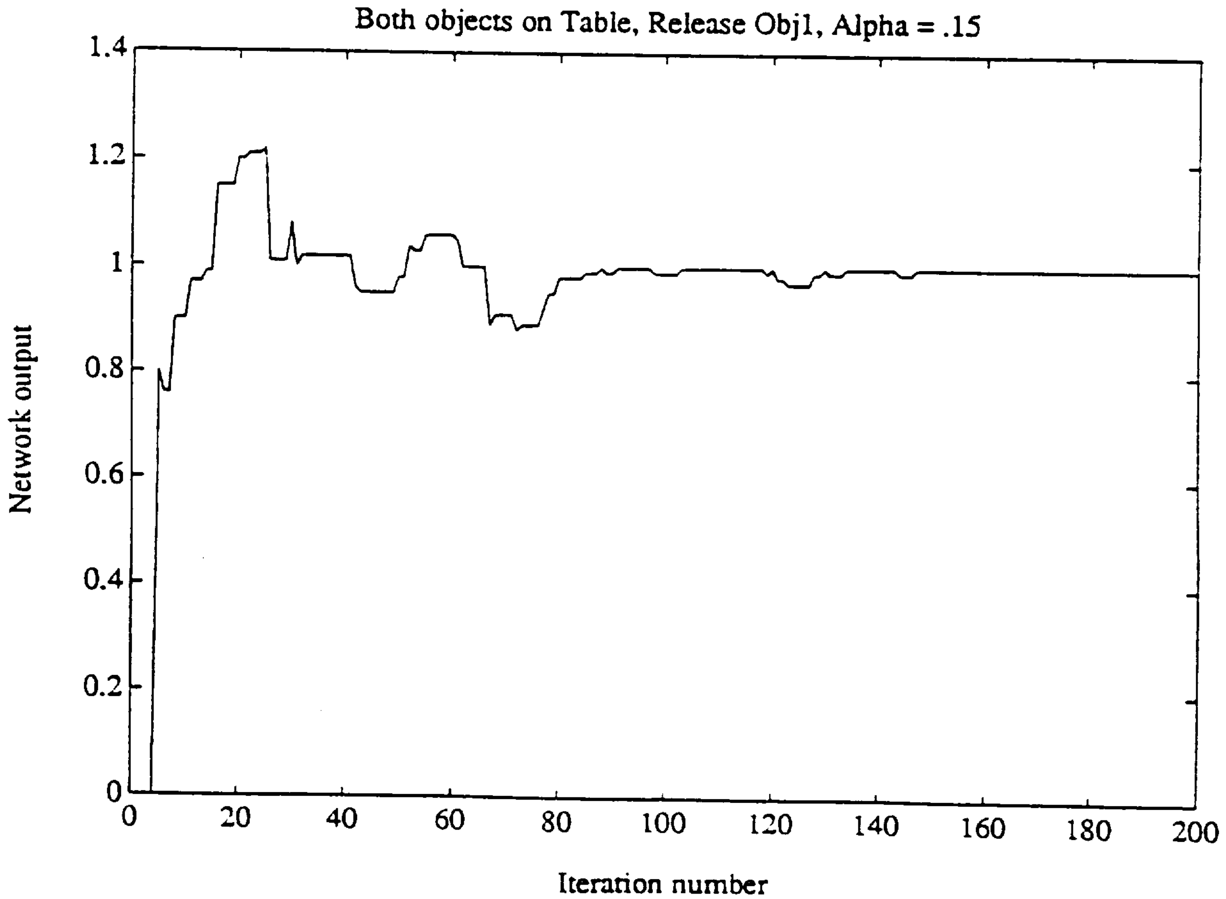


Figure 3.5b

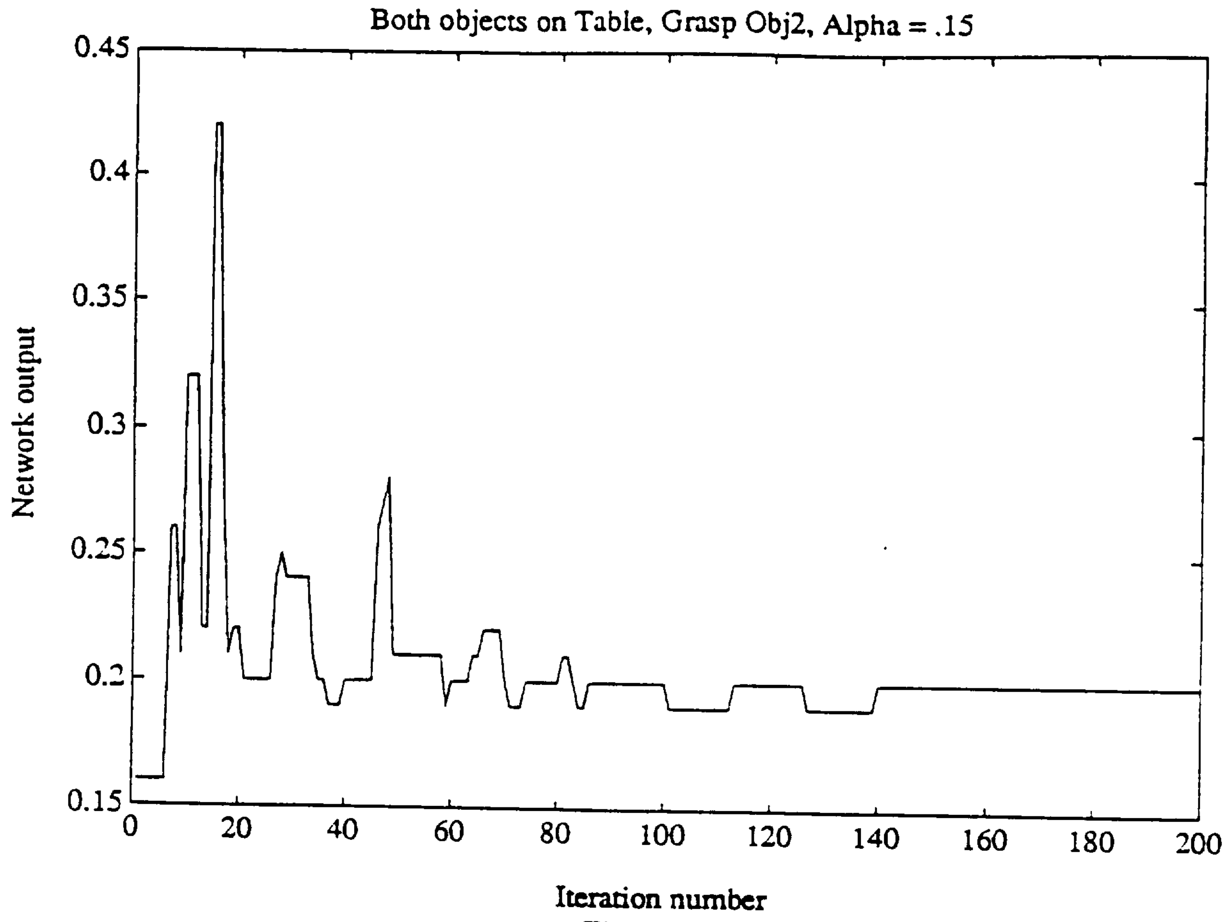


Figure 3.5c

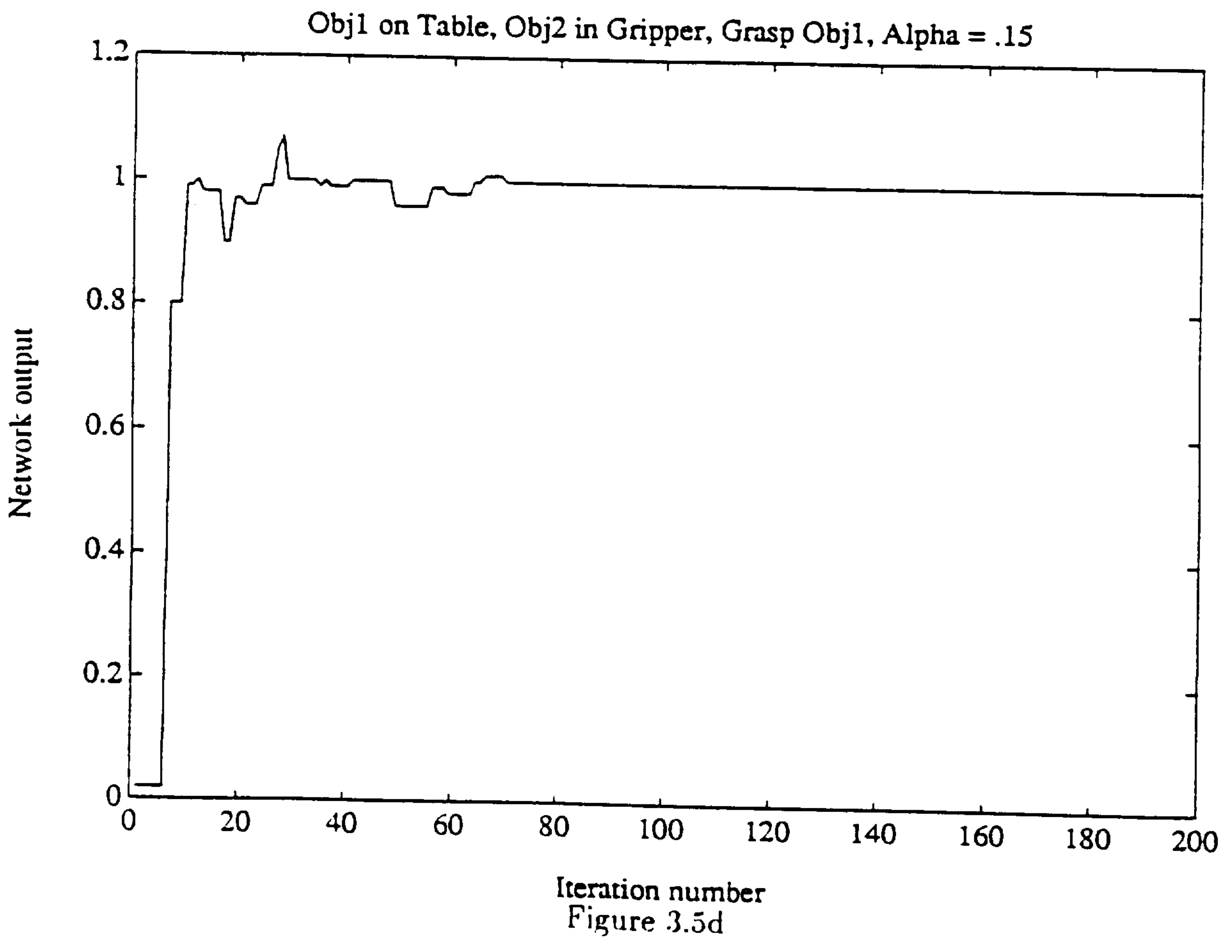


Figure 3.5d

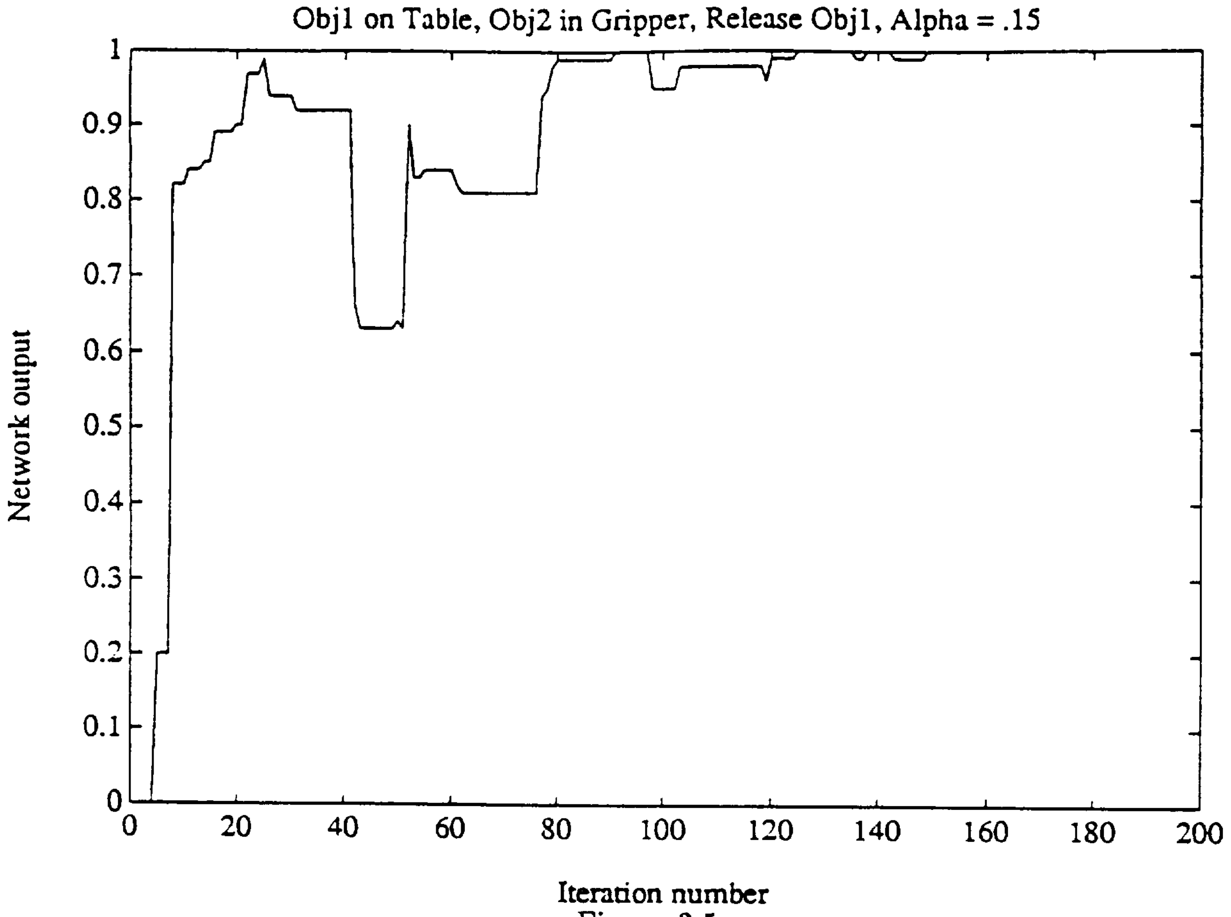


Figure 3.5e

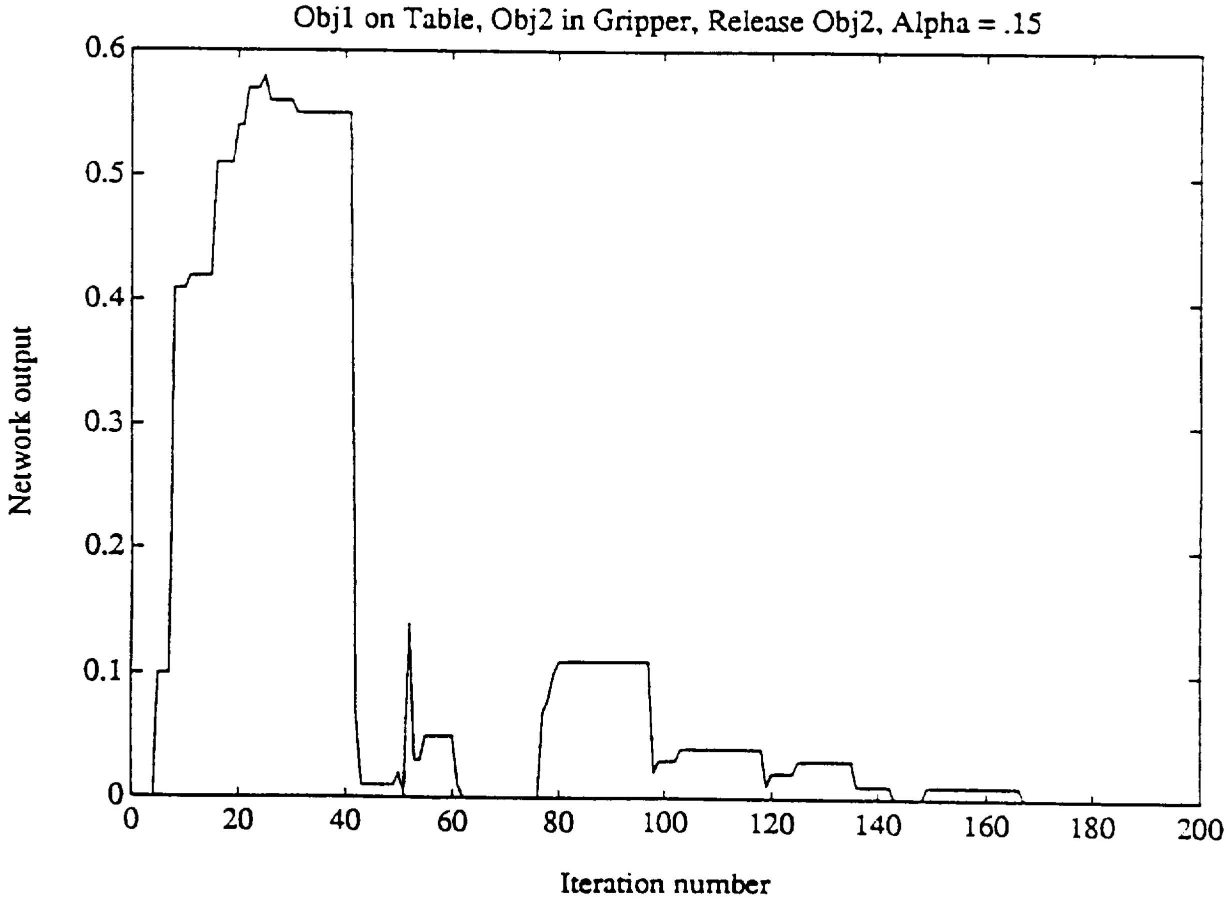


Figure 3.5f

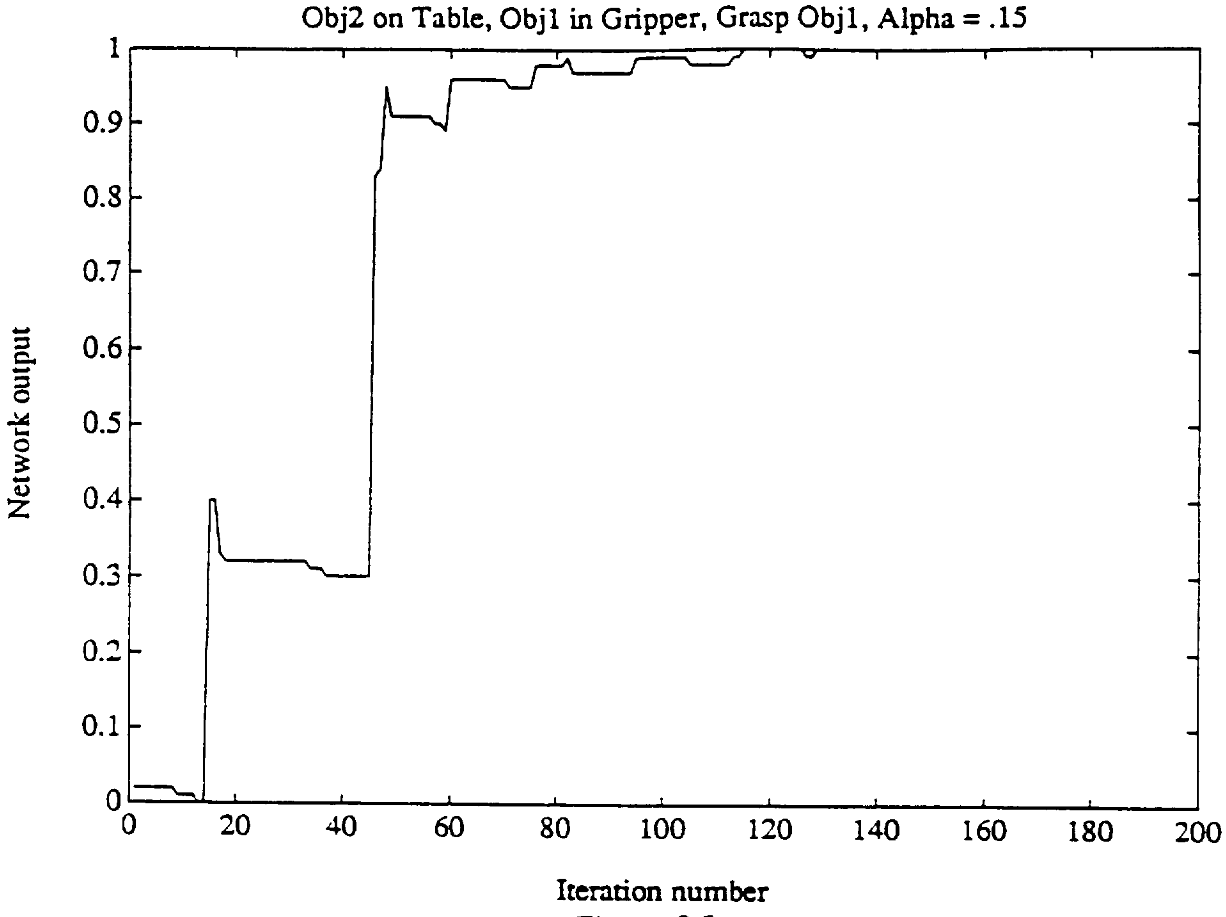


Figure 3.5g

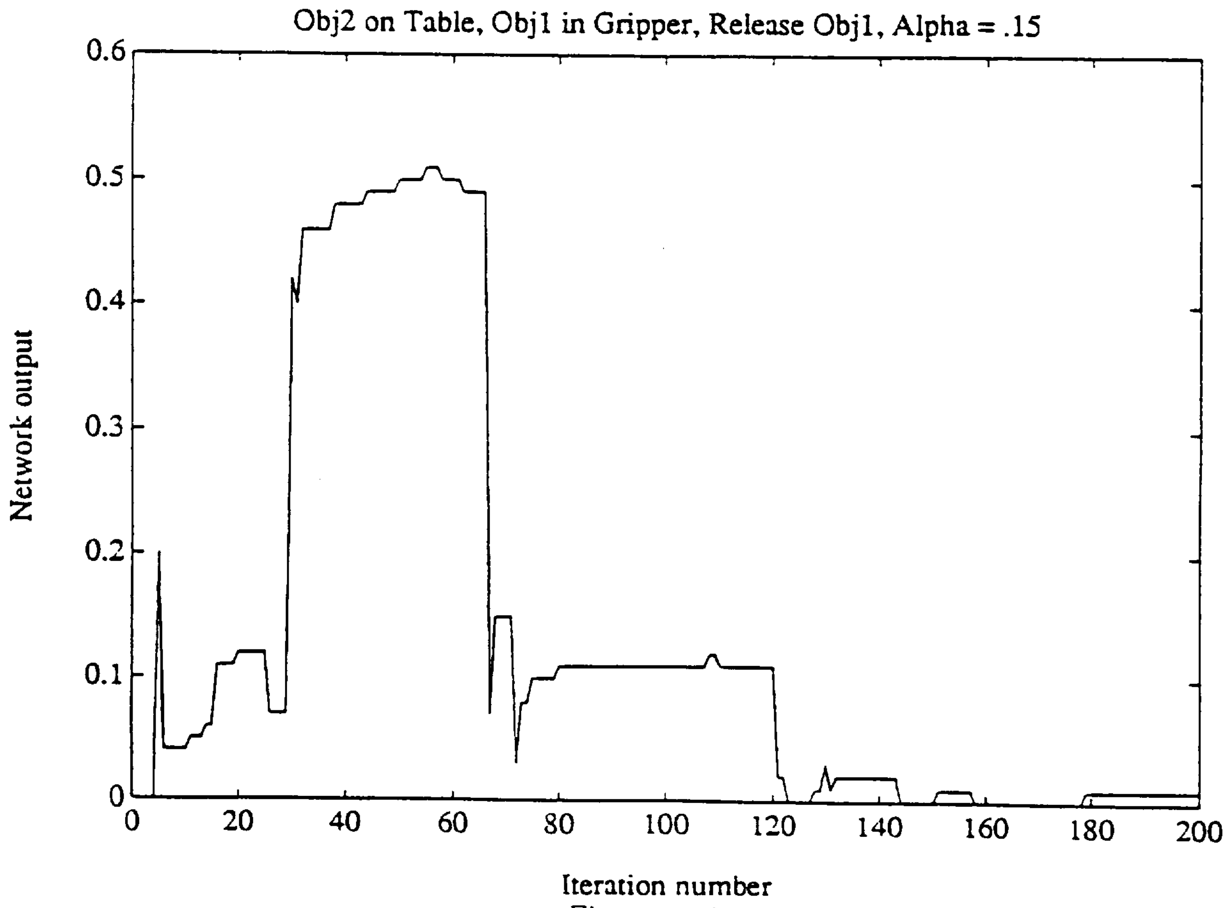


Figure 3.5h

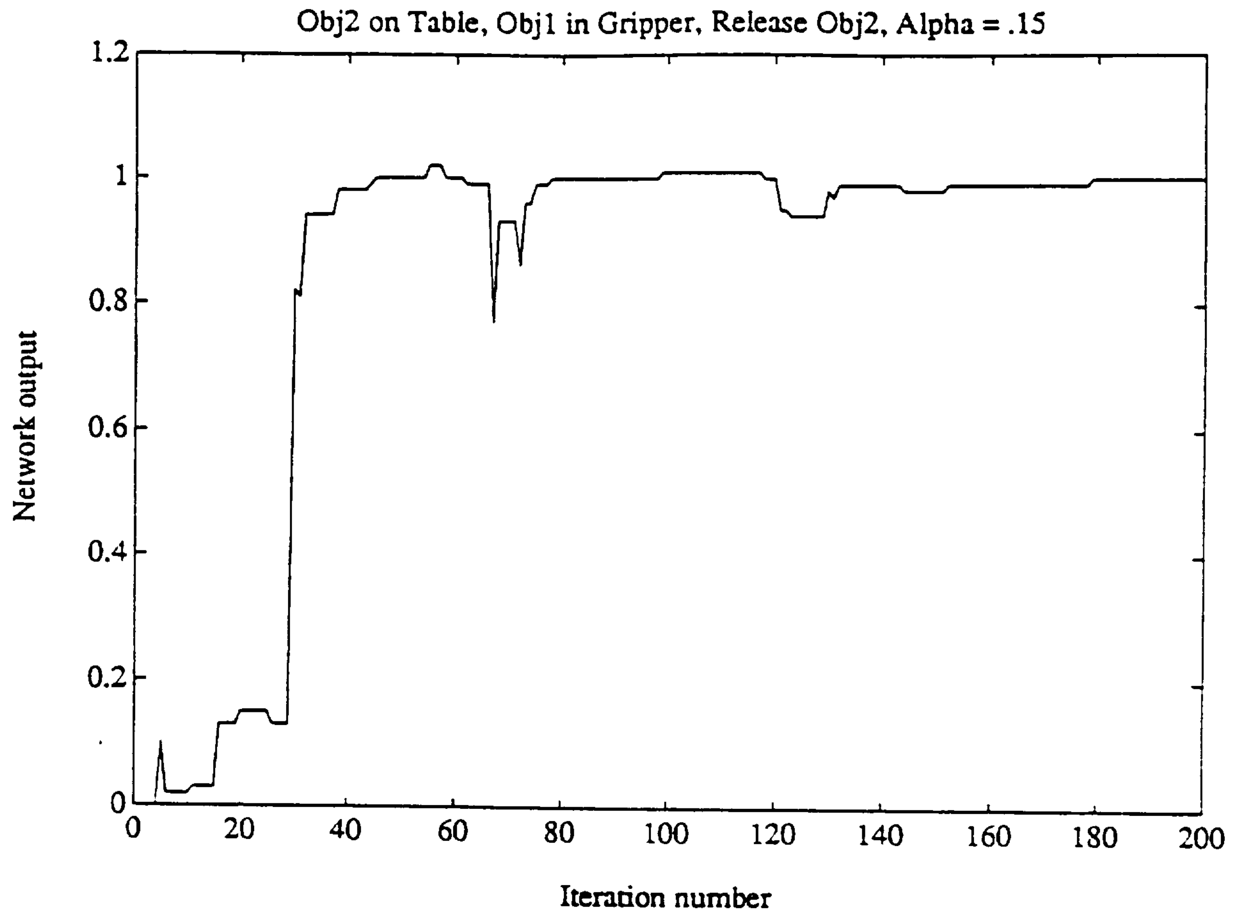
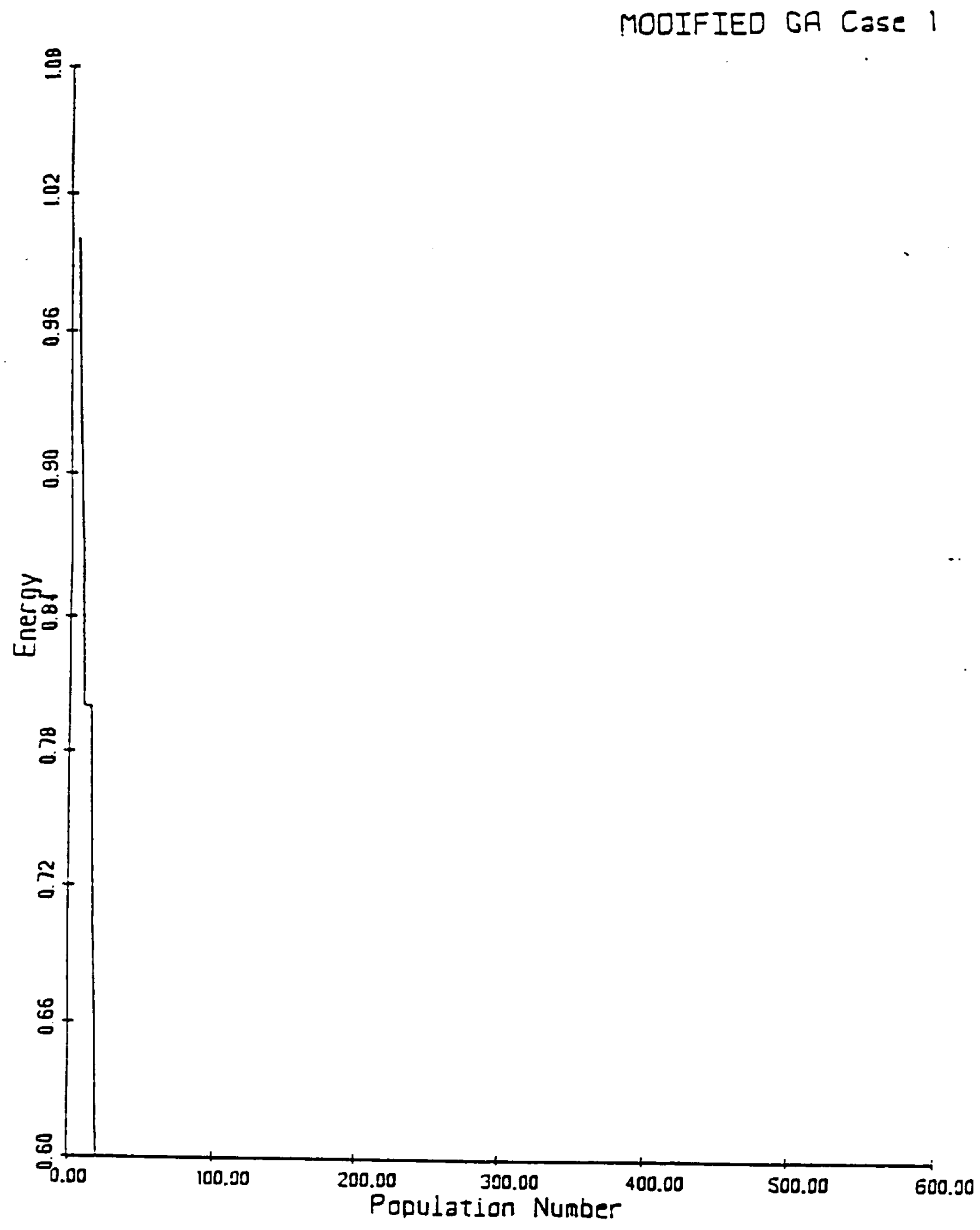

Obj2 on Table, Obj1 in Gripper, Release Obj2, Alpha = .15
Network output
Iteration number
0
0.2
0.4
0.6
0.8
1
1.2
0
20
40
60
80
100
120
140
160
180
200

Figure 3.5i



Best Case

Figure 3.6a

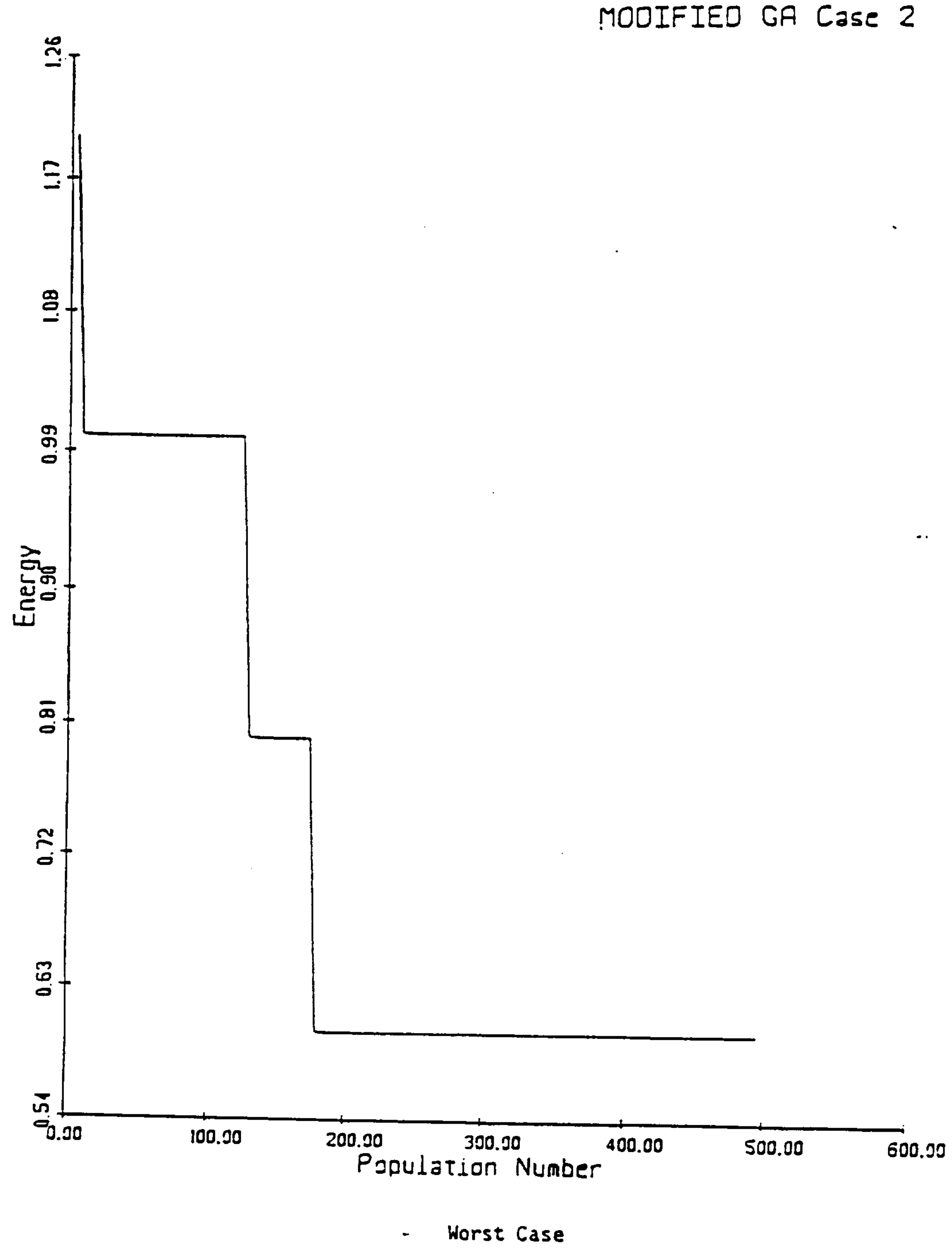
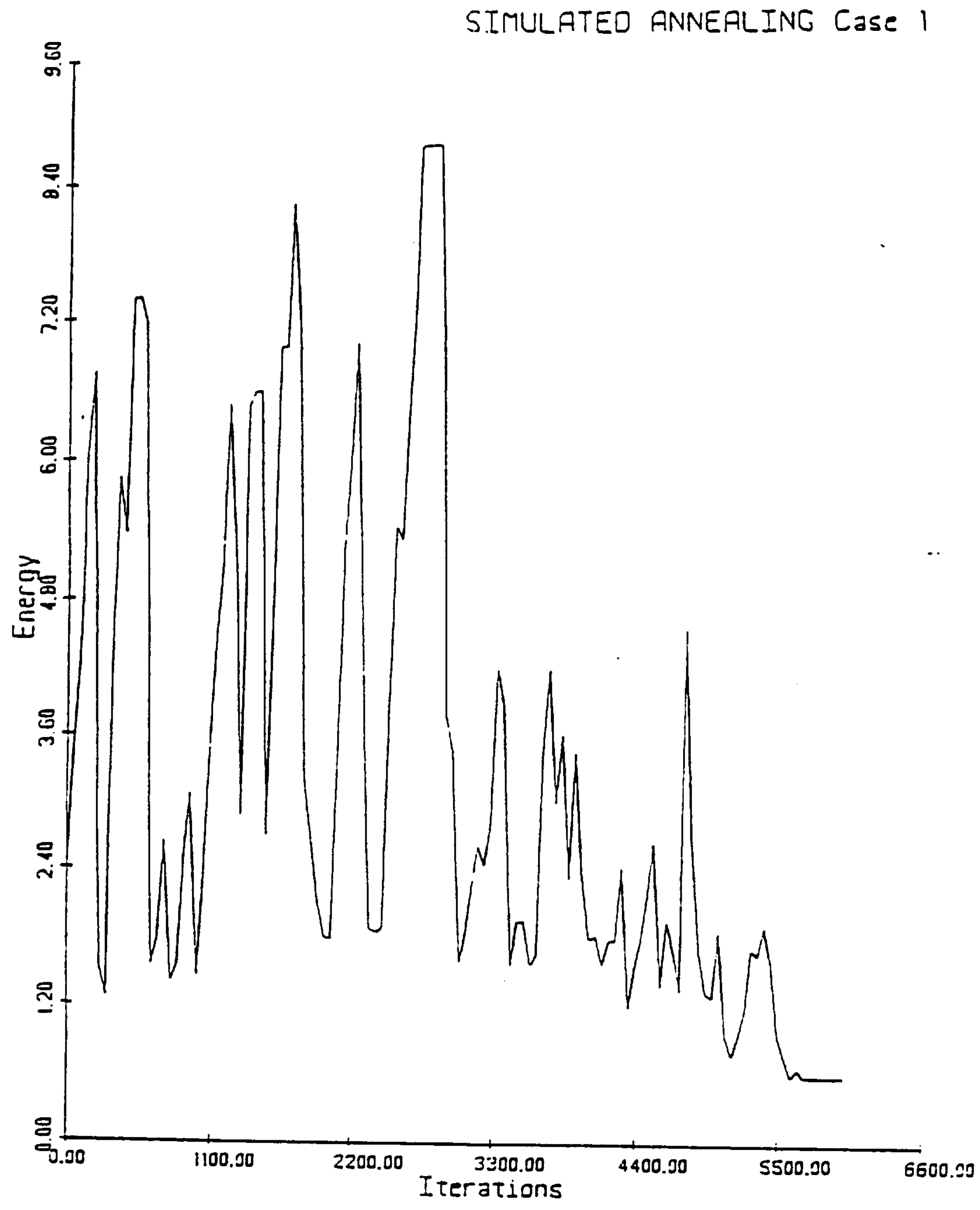
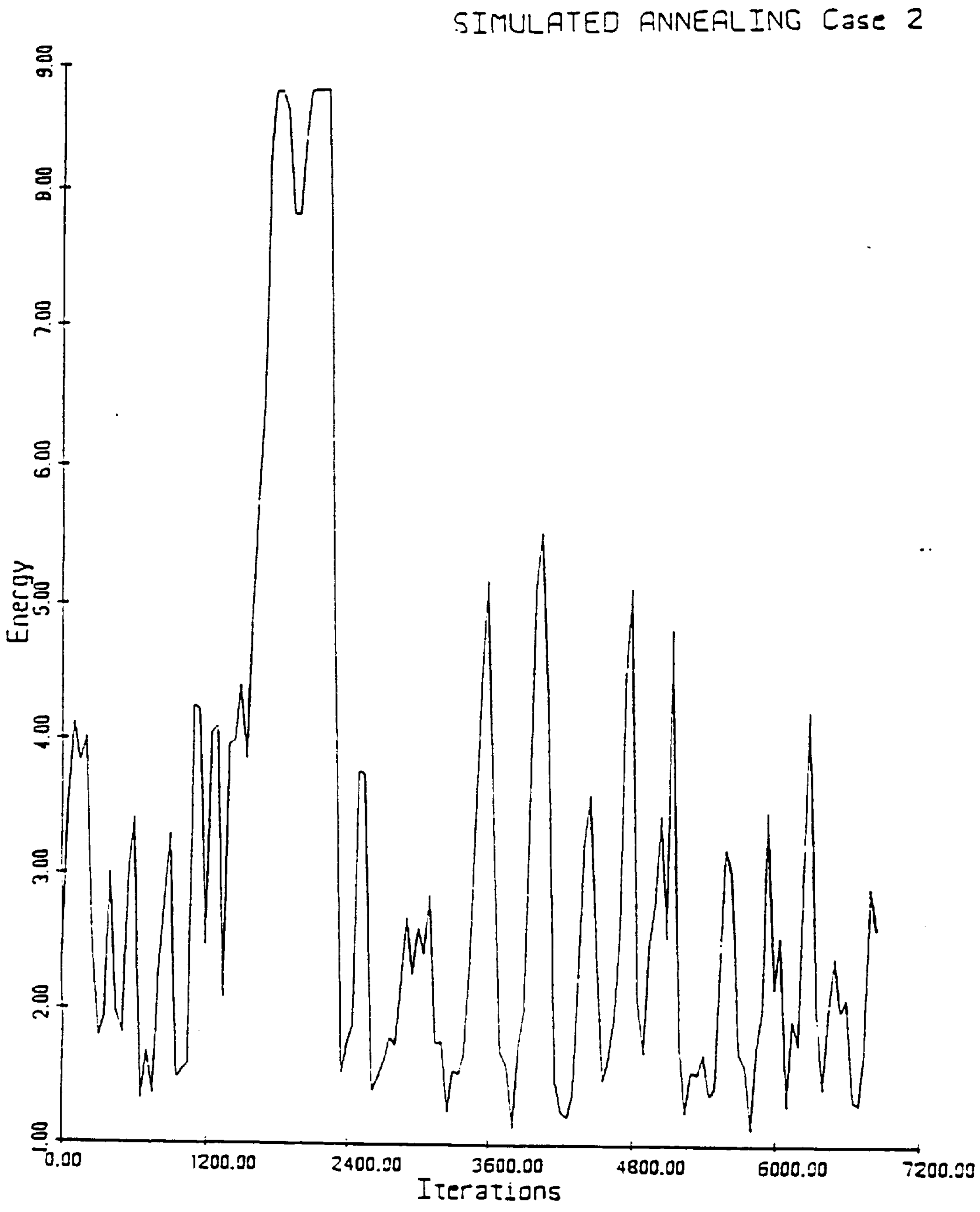

MODIFIED GA Case 2
1.26
1.17
1.08
0.99
0.90
0.81
0.72
0.63
0.54
Energy
0.00
100.00
200.00
300.00
400.00
500.00
600.00
Population Number
- Worst Case

Figure 3.6b



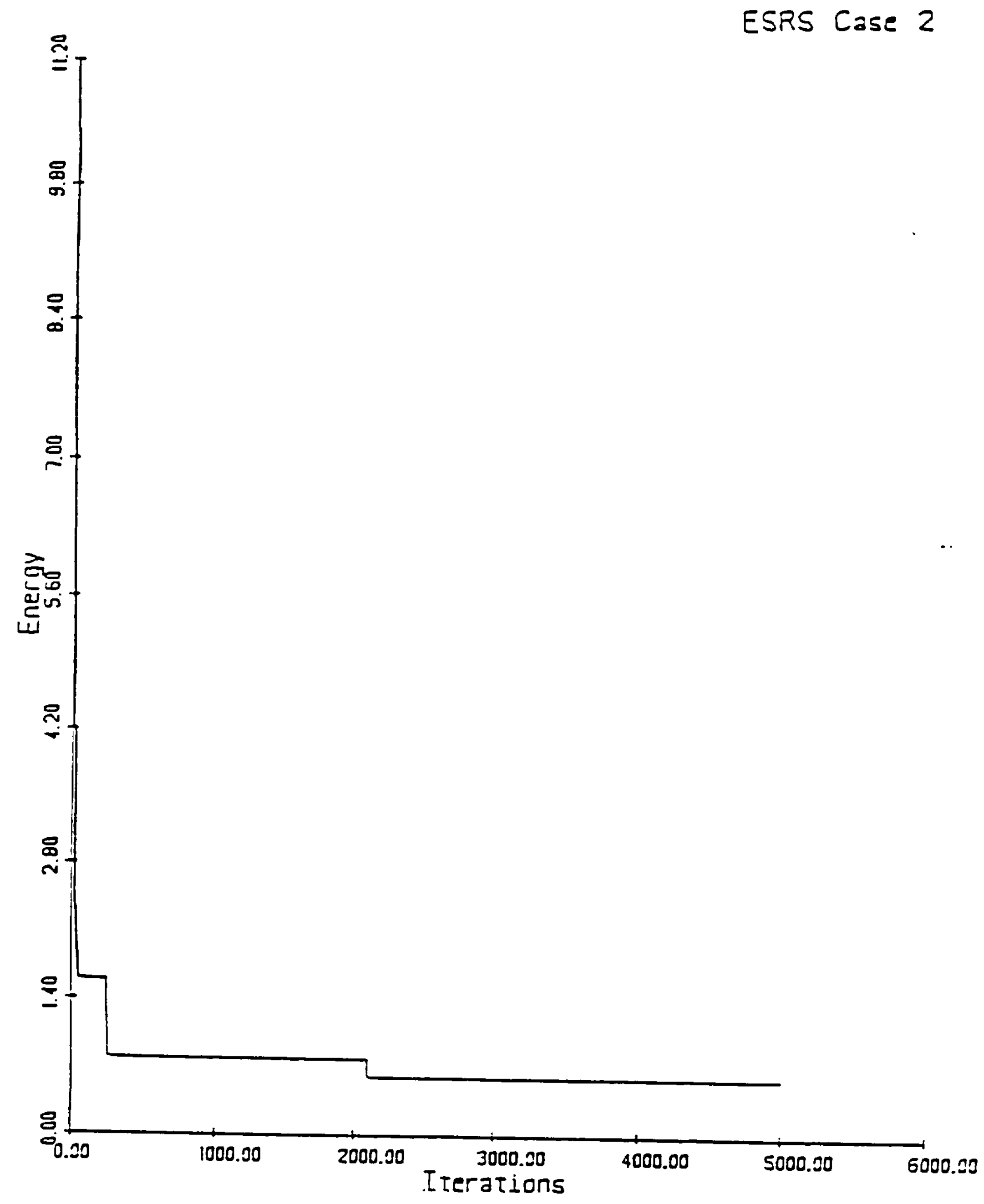
Best Case

Figure 3.6c



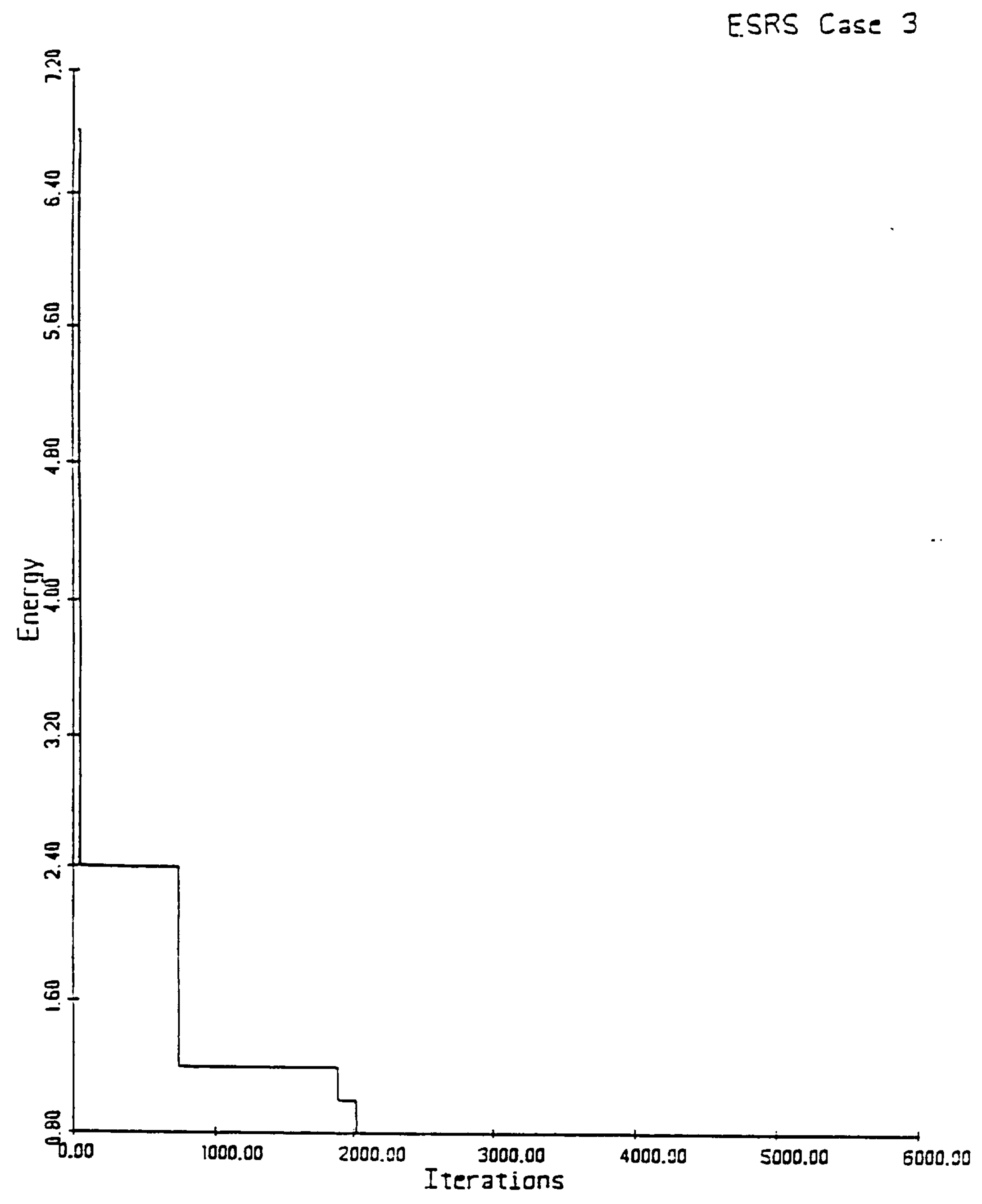
Worst Case

Figure 3.6d



Best Case

Figure 3.6e



Worst Case

Figure 3.6f